A MODEL DRIVEN ARCHITECTURE BASED APPROACH FOR DEVELOPING MULTI-AGENT SYSTEMS

A thesis submitted in partial fulfilment of the requirements for the

Degree

of Master of Science in Computer Science

in the University of Canterbury

by Di Zhou

University of Canterbury

2008

Contents

List of Figures

List of Tables

Acknowledgments

The author wishes to express sincere appreciation to Dr. Richard Pascoe for his constant valuable support, guidance and endless patience throughout this research project. In addition, special thanks to Dr. Warwick Irwin and Roger Jarquin from Jade Software Corporation for their advice and feedback on the drafts of this thesis. Thanks also to the members of the JMT department of the Jade Software Cooperation for their valuable input.

Abstract

The research described in this thesis is an attempt to utilize the Model Driven Architecture for semi-automatically developing a prototype Multi-Agent System to support the management of a real container terminal.

Agent technology has been increasingly applied in Transport Logistics and seems to be a viable solution to support the container terminal management. Thus, from the user point of view, the focus of this research is to investigate the applicability of Multi-Agent Systems to assist the container terminal's decision makers in improving the container terminal productivity, which is often measured in terms of the productivity of cranes. A prototype Multi-Agent System has been developed to evaluate and compare a set of proposed vehicle dispatching strategies, which are a collection of rules that a vehicle (e.g. straddle carrier) uses to decide the priority of serving the working cranes. Employing an appropriate dispatching strategy may greatly improve the efficiency of vehicle allocation to the working cranes, so as to increase the utilization of cranes which directly enhance the container terminal productivity. In order to investigate the applicability of the Multi-Agent System for supporting the container terminal management, experiments have been conducted in a variety of real-world scenarios. The experiment results have revealed that Multi-Agent Systems are applicable to assist container terminal decision makers in evaluating operating strategies.

On the other hand, from the developer point of view, the author investigates how to apply the Model Driven Architecture to agent technologies, providing a partially automated support for the derivation of Multi-Agent System implementation from the agent-oriented design, independently from the target implementation platforms. The Model Driven Architecture approach studied in this research is a model-driven software development process that explicitly separates models at three different levels of abstraction: platform independent models, platform specific models, and implementation models. In contrast to the conventional code-centric software development, the Model Driven Architecture based software development uses models as the primary engineering artifacts. The adopted development approach is to take a high-level abstraction model of a system and transform it into a set of platform specific models, each of which is in turn transformed into the corresponding implementation. Transformations between models are automatically carried out by a set of transformation tools. The experience of using the Model Driven Architecture for the development of the prototype Multi-Agent System has revealed the following benefits:

(a) automated transformations between models increase software productivity; (b) separating the high-level specification of the system from the underlying implementation technology improves the portability of the system's high-level abstraction model; (c) strong separation of concerns, guaranteed consistency between models, and automatic generation of source code minimize future software maintenance effort.

Nomenclature

ACL Agent Communication Language is based on speech act theory: messages are actions, or communicative acts, as they are intended to perform some action by virtue of being sent.

Agent An agent is an autonomous entity that has the unique characteristics: autonomy, social ability, reactivity and pro-activeness.

AGV An automatic guided vehicle is a mobile robot used in industrial applications to move materials around a manufacturing facility or a warehouse.

AML Agent Modeling Language is a semi-formal visual modeling language developed by Whitestein Technologies for specifying, modeling and documenting MASs.

BDI Belief-Desire-Intention is one of the most influential and successful agent theories, which was originally conceived by Bratman as a philosophical theory of human practical reasoning.

BDI Agent A rational decision-making system that comprises a BDI interpreter, an input event queue and the four key data structures: Beliefs, Desires, Intentions, and Plans.

Beliefs Information that an agent has about the current state of the environment within which it is situated.

Containers Containers are metal boxes which can be classified, based on the different sizes that are all measured in feet, as twenty foot equivalent unit containers or forty foot equivalent unit containers.

Desires The objective environment states (i.e. future world states) that an agent tries to accomplish, although the agent may not be able to fully achieve them.

Executable UML The MDA approach to specify the complete behavior of a system in the high-level abstraction model, which is fully automatically translated into the executable code by an executable UML compiler

FIPA Foundation for Intelligent Physical Agents is an IEEE Computer Society standards organization that promotesagent-based technology and the interoperability of its standards withother technologies.

FIPA-CNP FIPA Contract Net Interaction Protocol is a domain-independent protocol used for negotiations between one initiator agent and an arbitrary number of participants.

FIPA-RP FIPA Request Interaction Protocol is a domain-independent protocol that manages the interaction between one initiator agent and one participant agent.

Implementation A model that consists of all the information needed to construct a running system

Intentions A subset of desires that an agent has committed to pursuing in the current state of the environment.

JADE JADE is an object-oriented software development platform that includes an object oriented database management system (OODBMS).

JADE Profile A UML profile that provides constructs for modeling data to be persisted in JADE.

JADE-Java API A framework developed by Jade Software Cooperation to provide all of the functionality required to add object persistence to a Java application.

Jadex Jadex is a Belief Desire Intention reasoning engine that allows for programming intelligent software agents in XML and Java.

Jadex Profile A UML profile that provides constructs for modeling Jadex agents.

JMT Jade Master Terminal is a cargo management system which delivers sophisticated ship planning and yard control using radio telemetry.

Mappings Transformation rules for automatically transforming the elements, such as stereotypes and tagged values, defined in the platform independent UML profile into the corresponding elements provided with the platform specific UML profile.

MAS Multi-Agent System is a community of autonomous agents that operate to achieve individual goal(s) whilst collaborating and negotiating with each other to fulfill an overall goal.

MDA Model Driven Architecture is a model-driven approach to designing and developing portable, interoperable, reusable software components and data models

Mental Attributes Refering to the four key data structures of an BDI agent: Beliefs, Desires, Intentions, and Plans.

Model A description of (part of) a system written in a well-defined language which is suitable for automated interpretation by a computer.

Model Transformation The process of converting a model (e.g. the PIM) into another model (e.g. a PSM) using a collection of mappings.

MOF Meta Object Facility is an OMG standard that defines the language to define modeling languages.

PIM Platform-Independent Model is a high-level abstraction model of a system that is created independently of any particular implementation technology.

Plans Information about the means of pursuing intentions and options (i.e. possible actions) available to an agent.

PSM Platform-Specific Model is a concrete model that is constructed by adding to the PIM the details of how an underlying implementation technology is to be used in the system implementation

QC A quay crane is employed on the quay to load/ discharge containers to/from the vessel.

QiQu An open source framework to support the MDA approach.

SC A straddle carrier is a man-driven machine that are used to transport containers between the wharf and the yard.

StarUML An open source software modeling tool that supports UML to develop a UML/MDA platform running on a Win32 platform

TEU The Twenty-foot Equivalent Unit is the standardized unit of cargo capacity often used to describe the capacity of container ships and container terminals. The most common dimensions for a 1TEU container are 20 feet long x 8 feet wide x 8.5 feet high.

UML Unified Modeling Language is a modeling language that can greatly help in developing software systems by raising the level of abstraction from programming languages to models.

UML Profile Introduced to attach additional semantics, properties and constraints to existing UML elements using stereotypes, tagged values and constraints respectively.

Vehicle Dispatching Strategie A collection of rules that a vehicle (e.g. straddle carrier) uses to decide the priority of serving the working cranes.

Voyage A visit of a vessel to a terminal.

XMI XML Metadat Interchange is a standard way, defined using the MOF, to generate an interchange format based on XML for models defined in a modeling language whose metamodel is described in the MOF.

Chapter 1

Introduction

In this thesis is described the use of Model Driven Architecture (MDA) based approach to develop a prototype Multi-Agent System (MAS) that is applied to support the management of a real container terminal. The research project, which is motivated by real-world industry problems, is a collaborative project between the New Zealand ICT Innovation Institute[1] and Jade Software Cooperation[2]. Thus, the objectives of the research can be viewed from two perspectives. From the user point of view, the focus of this research is to investigate the applicability of MASs in assisting the decision makers at Port Otago in improving the productivity of their container terminal. A prototype MAS has been developed to achieve this objective. From the developer point of view, the author focuses on the development of MASs using the MDA [18] based approach within the context provided by the JADE platform. JADE is an object-oriented software development platform that includes an object oriented database management system (OODBMS). In order to accomplish this objective, a set of transformation rules specifying the development of the prototype MAS have been defined using existing Computer-Aided Software Engineering (CASE) tools that support the MDA approach. The research objectives, questions and methodology are discussed in more detail later in this chapter.

In Chapter 2 is presented the background of this research. Worldwide container trading has been growing rapidly (about 7-9% per year for over the last two decades [70]) since containers were introduced into the market for cargo transportation in the early 1960s. The steady growth in number of containers and size of container vessels may cause a longer berthing time in a port terminal, resulting in a higher cost (U.S. $1000 for each hour a ship spends in a port terminal [7]). In order to minimize the ship turn-around time (i.e. to minimize

[1] http://www.uci3.canterbury.ac.nz/
[2] http://www.jadeworld.com/

costs), a possible solution is to enhance container terminal productivity which is often measured in terms of the productivity of quay cranes (QCs) [50]. Efficient allocation of vehicles to the working QCs during container discharging/loading operations may greatly increase the QCs' productivity which directly improves the container terminal productivity.

Agent technology has been increasingly applied to support the management and planning activities associated with Transport Logistics [47]. A MAS consists of a number of goal-directed autonomous agents that are able to perceive their environment and react to changes. The unique characteristics of agents (i.e. *autonomy*, *social ability*, *reactivity* and *pro-activeness*; see Section 2.1 for more detail) lead the agent technology to be a promising solution to the distributed, dynamic, and complex container terminal management problem [5].

As an outcome of this research, a prototype MAS described in Chapter 3 has been developed to assist the container terminal's decision makers in enhancing container unloading operations through better utilization of the available machines such as QCs and straddle carriers (SCs). In particular, the MAS is used to compare the performance of proposed SC dispatching strategies. In this research, a SC dispatching strategy is a set of rules that a SC uses to decide the priority of serving the working QCs assigned to a container vessel. For a given vehicle dispatching strategy, the prototype MAS is applied to find the number of SCs that should be deployed for a container vessel during the container discharging operation, so as to minimize the idle times of working cranes caused by overflows of the limited storage buffer below them. Furthermore, this prototype MAS interoperates with JADE 6.2 through the new JADE-Java API to store all information represented as objects to help the MAS achieve its goal.

In Chapter 4 is described experiments carried out in order to test the applicability of the MAS in the application domain of logistics. An objective of the experiments is to compare the performance of the proposed SC dispatching strategies in a variety of real-world scenarios that vary in terms of ship sizes and numbers of QCs. Realistic input data of the experiments is taken from Port Otago's Jade Master Terminal (JMT) application, which delivers sophisticated ship planning and yard control using radio telemetry. JMT stores all the real-time information on container flows and various available resources, and this data is used in the experiments.

A MDA based approach can simplify the development of MASs by partially automating the engineering process. In Chapter 5 is explained in detail how the MDA based approach is applied in this research to semi-automatically develop the prototype MAS. In Figure 1.1 is depicted the approach involving various models at different levels of abstraction as well as the corresponding transformations used to generate them. Creating the platform independent business

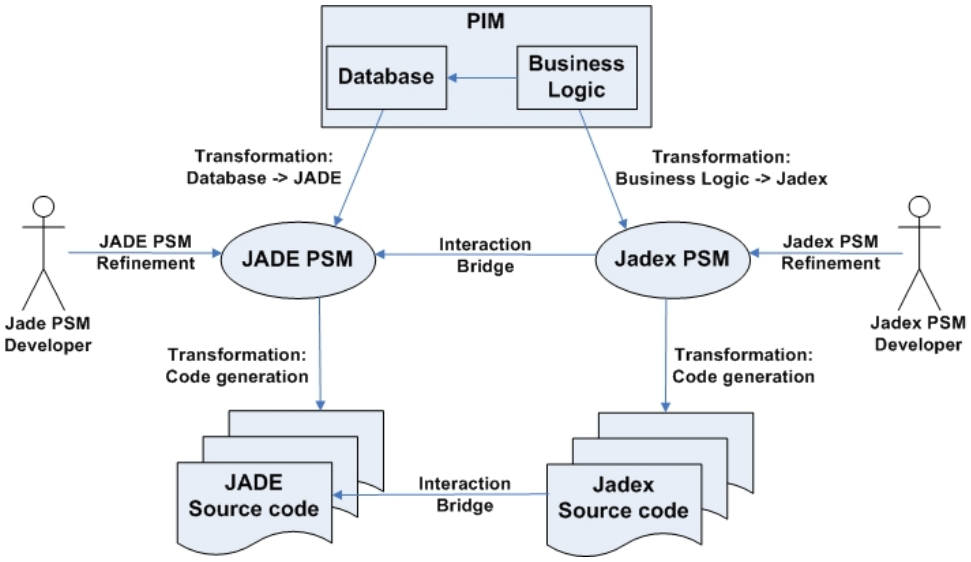

Figure 1.1: MDA process for the MAS application.

model of the prototype MAS (i.e. the PIM) is the starting point of the MDA process. The PIM usually comprises multiple components (e.g. database and business logic) each of which is under a separate control. As each component of the PIM is transformed into a separate platform specific model targeting a different implementation technology, an interaction bridge is then constructed to enable the interoperability between the distinct PSMs. Each transformed PSM that may be refined manually by developers is finally transformed into the corresponding source code.

This thesis is concluded in Chapter 6 with a summary of what has been achieved, the answers to the research questions of this thesis, and finally a discussion of possible future research.

1.1 Research Aims and Objectives

The aim of this research is to utilize the MDA based approach for the development of MASs within the application domain of Logistics and in particular, but not limited to, the managed complex logistical activities within a container terminal. In order to realize the aim, the following objectives have to be fulfilled.

The first objective of this research is to develop a prototype MAS to assist the operation manager at port of Otago in enhancing container unloading operations through better utilization of the available resources. The specific goals of the output MAS are described as the following:

1. Evaluate and compare the effectiveness of different vehicle dispatching

strategies in a variety of real-world operation scenarios based upon:

- The data gathered from Port Otago's JMT application.
- The minimum number of vehicles to be deployed under a given vehicle dispatching strategy so as to minimize the working cranes' idle times, which are caused by overflows of the limited storage buffer below them.

The second research objective is to enhance the process of engineering MASs in an automatic fashion based on the MDA approach. Kleppe et al. [22] state that the key challenge in the MDA approach is to develop transformations for transforming the system high-level abstraction model into a set of related platform specific models and further for generating the implementation. According to their statement, a list of specific goals to be achieved by this research are:

1. Develop the high-level abstraction model of the prototype MAS.
2. Develop a set of transformation rules for realizing the abstract model of the prototype MAS as a set of implementation technology specific models.
3. Develop a set of transformation rules for transforming the set of implementation technology specific models into the corresponding implementation.
4. Develop a set of interaction bridges by which the distinct implementation technology specific models are to interoperate despite differences in programming languages and execution platforms.

1.2 Research Questions

The research questions are:

1. What motivates the use of MDA in the development of software systems?
2. How to apply MDA to provide a partially automated support for the derivation of MAS implementation from the agent-oriented design?
   (a) How to define the high-level abstraction model of the MAS?
   (b) How to transform the high-level abstraction model of the MAS into a set of implementation technology specific models?
   (c) How to transform the set of implementation technology specific models into the corresponding implementation?
   (d) How to enable the distinct implementation technology specific models to interoperate despite differences in programming languages and execution platforms?

1.3 Research Methodology

This section presents the following three methods through which the research proceeded.

1. *Literature review*: An overview of prior research focusing on improving container terminal productivity by the efficient use of the available resources is provided in Section 2.4.1.3. Section 2.4.2.2 provides a summary of research projects using agent-based technologies to solve problems in transport and logistics domains. Some work related to vehicle dispatching strategies for improving container terminal productivity was surveyed and is presented in Section 2.4.2.3.

2. *Interview*: An open-ended discussion was conducted with the operation manager and the ship controller at port Otago in order to identify the issues and opportunities related to the enhancement of port terminal productivity. A number of functionalities that are desirable to meet future challenges are summarized from the interview and described in Appendix A.

3. *Experiments*: As an outcome of the research, a prototype MAS has been developed in a semi-automatic fashion using a MDA based approach. A set of experiments were conducted in order to test the applicability of the MAS to evaluate and compare the performance of proposed vehicle dispatching strategies. More specifically, the experimental objective is to have different number of SCs in each scenario (i.e. a voyage) to investigate the performance of the three proposed SC dispatching strategies during the container discharging operation. A variety of real-world operation scenarios with realistic input data from Port Otago are used to strengthen the experiments in order to correctly and comprehensively compare the effectiveness of the proposed dispatching strategies.

Chapter 2

Research Background

In this chapter is presented the research background. Firstly, a definition of MASs is provided in Section 2.1. Secondly, in Section 2.2 is summarized the agent architecture adopted in this research to develop rational agents. Thirdly, in Section 2.3 is described a set of standard domain-independent protocols, which enable agents to communicate. Subsequently, the use of MASs in container terminals is discussed in Section 2.4. In particular, the focus is on the use of MASs to evaluate and compare different vehicle dispatching strategies rather than algorithms or other means employed to enhance container terminal productivity. Finally, in Section 2.5 is given an overview of the MDA adopted to facilitate the development of MASs and a summary of CASE tools that support the MDA approach.

2.1 Multi-Agent System Definition

Although the literature offers a variety of agent definitions, there is none that is standardized and universally accepted by all researchers [62]. Among a variety of agent definitions in literature (e.g. [25], [26], [56], and [57]), this research adopts the one defined by Woolbridge and Jennings [26, p.2]:

"*... a hardware or (more usually) software-based computer system that enjoys the following properties:*

- *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;*
- *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;*
- *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents,*

the internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;

- *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative."*

In order to gain a better understanding of what an agent is and how they differ from programs in general, it may be useful to distinguish agents from expert systems. Agents differ from expert systems in the following respects [24]:

- Two characteristics that make agents different from expert systems are *reactivity* and *social ability*. Unlike expert systems, which usually interact with their environments via users, agents are able to notice (e.g. via sensors) changes occurring in the environment within which they are situated, respond to those changes in a timely manner, and perhaps in turn also affect the environment. Furthermore, expert systems do not normally operate in real time. Neither do they usually collaborate and negotiate with each other to achieve an overall goal.

Based on agents' unique characteristics that are discussed above, a MAS can be defined as a community of such agents that operate to achieve individual goal(s) whilst collaborating and negotiating with each other to fulfill an overall goal.

2.2 Agent Architecture

Various agent architectures have been proposed to develop rational agents (e.g. [58], [66], [67] and [68]). This section explores the Belief-Desire-Intention (BDI) model, which is one of the most influential and successful agent theories [77]. The BDI model was originally conceived by Bratman as a philosophical theory of human practical reasoning [55]. Rao and Georgeff have formalized the original BDI model into a comprehensive family of BDI logics [58].

Rao and Georgeff 's BDI model is adopted in this research for developing rational agents. A BDI agent is a rational decision-making system that comprises a BDI interpreter, an input event queue and the four key data structures: *Beliefs*, *Desires*, *Intentions,* and *Plans* [54].

- Beliefs represent information that an agent has about the current state of the environment within which it is situated. Thus, beliefs are what an agent believes about the current state of the world. Furthermore, an agent's beliefs may not necessarily be true and probably will change in the future.

- Desires represent the objective environment states (i.e. future world states) that an agent tries to accomplish, although the agent may not be able to fully achieve them. They are the motivational states of an agent.
- Intentions represent a subset of desires that an agent has committed to pursuing in the current state of the environment. Intentions, the chosen desires, are the deliberative states of an agent.
- Plans represent information about the means of pursuing intentions and options (i.e. possible actions) available to an agent. They define courses of actions that an agent will perform in order to pursue chosen desires. A plan may contain not only a set of primitive actions but also sub-goals which have to be achieved for plan execution to be successful.

A BDI agent's practical reasoning process is carried out in two phases: *deliberation* and *means-end reasoning* [63]. The deliberation phase is responsible for deciding an agent's intentions (i.e. a consistent subset of goals) based on the agent's current beliefs, desires and intentions. The means-end reasoning phase is responsible for determining how an agent is going to achieve the chosen desires by executing certain plans.

2.3 Agent Communication

The Foundation for Intelligent Physical Agents (FIPA) has standardized several domain-independent protocols [72], which enable agents to communicate. Among those standard protocols, the *FIPA Request Interaction Protocol (RP)* and the *FIPA Contract Net Interaction Protocol (CNP)* are used more frequently in the prototype MAS and thus described in detail in this section. Moreover, messages communicated among agents are based on the FIPA Agent Communication Language (FIPA-ACL[89]) standard. The FIPA-ACL is based on speech act theory: messages are actions, or communicative acts, as they are intended to perform some action by virtue of being sent.

The FIPA-RP, depicted in Figure 2.1, manages the interaction between one initiator and one participant. The initiator sends a request message to the participant to ask it to perform some action. The participant processes the request and then replies an optional agree message or a refuse message to inform the initiator its decision. If the participant has agreed, it then performs the required action. Once the action has been completed, the participant communicates the action result as either a failure or an inform message to the initiator. The inform message may contain the result of the action execution.

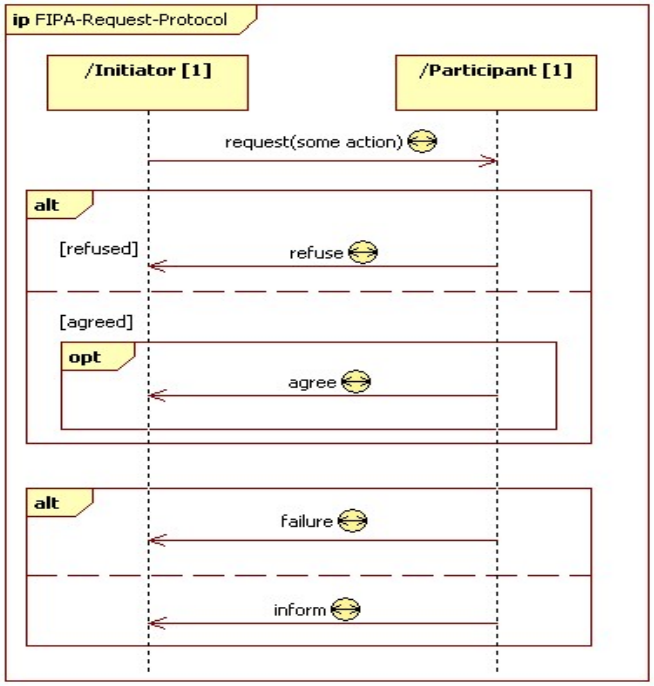

Figure 2.1: A sequence diagram depicts the FIPA-RP.

One of the most important tasks is the collaboration and negotiation between agents. FIPA-CNP shown in Figure 2.2 is used for negotiations between one initiator and an arbitrary number of participants. To begin with, the initiator issues call for proposal (cfp) messages which specifies the task to m participants. Participants receiving the cfp message are potential contractors which are able to give n ($n \leq m$) responses before a given deadline has passed. If there are j ($j \leq n$) proponents who make proposals then there are $(n - j)$ refusers who refuse the cfp. Once the deadline passes, the initiator evaluates the received j proposals to select none, one or several proponents to perform the task. The k ($k \leq j$) rejected agents will be sent a reject-proposal message and the remaining $(j - k)$ accepted proponents will receive an accept-proposal message. Once each accepted proponent has completed the task, it sends an inform message to the initiator. If an error occurs, a failure message will be sent back to the initiator.

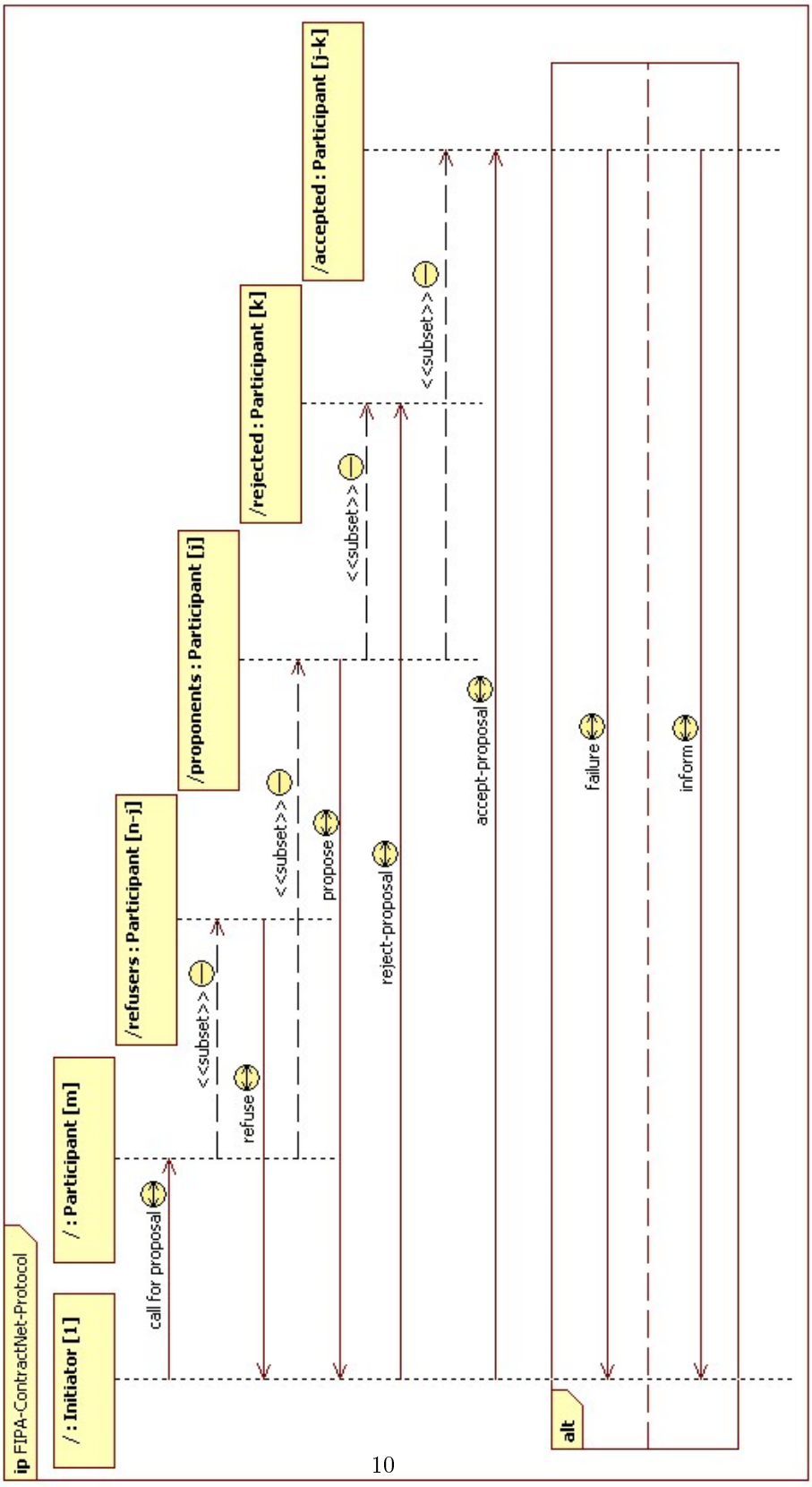

Figure 2.2: A sequence diagram describes the FIPA-CNP.

Figure 2.3: Picture of port Otago.

2.4 Use of Multi-Agent Systems in Container Terminals

Agent technology has been increasingly applied to support the management and planning activities associated with Transport Logistics [47]. Furthermore, agent technology is considered as a promising approach for developing applications of container terminal management [5]. In this Section is discussed the use of MASs to assist the container terminal decision makers in enhancing container unloading operations by improving utilization of the available resources (i.e. QCs, SCs).

To begin with, a general description of container terminals is given. Next is presented the vehicle allocation problem including a description of the problem, a summary of prior work into the problem and a discussion of the use of dispatching strategies as a possible solution to the problem. After that is discussed the use of MASs for evaluating the proposed dispatching strategies.

2.4.1 Container Terminal Overview

In this section is given an overview of a real maritime container terminal (see Figure 2.3) at Otago, New Zealand. Firstly, containers and associated handling equipments are described. Secondly, the characteristics of container terminal management problem is presented. Finally, a number of prior research projects focusing on improving container terminal productivity are summarized.

2.4.1.1 Containers and Handling Equipments

Containers

Figure 2.4: An example of quay crane.

Containers were introduced into the market for cargo transportation in the early 1960s [74]. They are metal boxes which can be classified, based on the different sizes that are all measured in *feet*, as twenty foot equivalent unit (TEU) containers ($20 \times 8 \times 8.5$ or $20 \times 8 \times 9.5$), or forty foot equivalent unit containers ($40 \times 8 \times 8.5$ or $40 \times 8 \times 9.5$) i.e. 2 TEUs, or slightly larger size boxes e.g. refrigerated containers.

Quay Cranes (QCs)

Upon arrival at the port terminal, the container vessel is assigned to a berth. Then a number of manually operated QCs (see Figure 2.4) will be allocated to discharge/load containers from/to the vessel. Normally two or three QCs are assigned to one container vessel in the case of Port Otago. Discharging operations are generally performed before loading operations. As container ships are partitioned into bays, each assigned QC is responsible to discharge and load all the target containers at a given bay. Only one QC can work at the given bay at one time and can move to another bay only after completing the current one. A safety distance (four working ship-bays) between crane movements, the so called crane clearance, must be maintained all the time to avoid cranes clashing. As the outside cranes move longitudinally along the ship, the middle crane may be closed down to prevent physical interference with another crane when they get too closed to each other. Two types of QCs can be distinguished: single-trolley cranes and dual-trolley cranes. Single-trolley cranes, which are conventionally employed at port terminals, can move one TEU container at one time whereas dual-trolley cranes can move up to two TEU containers simultaneously, the so called twin-lift mode.

Figure 2.5: An example of straddle carrier.

To discharge a container from a vessel, a QC picks up a container at the given ship-bay and sets it down at its feet area. Note that if a QC's feet area reaches its maximum capacity (i.e. three containers in the port of Otago), the QC has to pause the discharging operation until at least one of the unloaded containers is moved. In the container loading operation, a QC picks up a container at the feet area and sets it down at the target position of the given ship-bay.

Straddle Carriers (SCs)

Man-driven SCs (see Figure 2.5) are used to transport containers between the wharf and the yard. In addition to transporting containers, SCs are also able to stack containers in the yard, normally three containers high. All the QCs that are currently working on one ship are served by a number of SCs, which originally queue up at the assigned vehicle transfer area. In the case of a container discharging operation, when the first idle SC in the queue is assigned to a container discharged by a QC, the SC will move to one of three SC lanes between the two rows of two legs of the QC. After the SC picks up the unloaded container, it will transport the container to the target position in the yard. It then returns to the transfer area to wait for the next job assignment. In the case of a container loading operation, a SC travels from the assigned vehicle transfer area to the position of a given target container in the yard. Once it picks it up, it will carry it to the corresponding QC to load it onto the ship. If the feet area of the QC is full then the SC has to wait in the queue. As soon as a space is free, the SC can proceed and drop the container at the crane's feet area waiting

for the crane to load it onto the ship. The SC then returns to the yard to pick up the next container.

2.4.1.2 Container Terminal Management

The Container Terminal Management (CTM) problem is characterized as follows:

- The CTM system consists of a number of independent entities (e.g. QCs and SCs) each of which has its own goal(s) to achieve whilst interacting with each other to pursue a common goal. Decisions made within each system may directly affect operations of the others.
- It is very difficult or impossible to build a single centralized system to fulfill all the requirements of a container terminal due to its distributed nature.
- Everyday a large number of containers arrive and leave the container terminal leading to a dynamic environment. There are a number of factors that can cause uncertainties and varieties in a container terminal, for instance the weather condition, machine breakdown and work delay due to port congestion and/or lacking manpower. Moreover, the distribution of workload within each independent process tends to be unevenly over time. All these factors cause difficulties for the decision making.
- Container terminal is generally regarded as one of the most complex environments. [35]

2.4.1.3 Prior Research on Improving Container Terminal Management

The worldwide container trading has been growing at about 7-9% per year for over the last two decades [70]. The number of TEU containers has increased from 39 million in 1984 to over 356 million in 2004, and the size of container vessels has increased to almost 14,000 TEUs [33]. The rapid growth of containerization requires container terminal management to improve efficiency of terminal operations in order to make their container terminal more competitive over other terminals. Various problems related to the improvement of CTM have been studied and some of them are summarized as follows:

- Berth allocation problem [48] [49];
- Ship loading/unloading sequencing problem [28];
- QCs scheduling problem [29] [50]; and

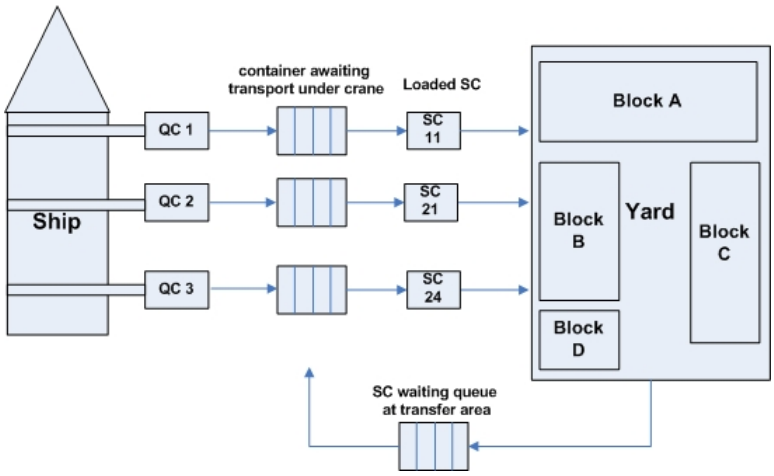

Figure 2.6: An example of container discharging process.

- Vehicle allocation problem [30] [51] [31].

2.4.2 Vehicle Allocation Problem

According to the interview with the operation manager and the ship controller at Port Otago, a number of issues (see Appendix A for more detail) have been identified as having an impact on container terminal productivity. The vehicle allocation problem has been selected to be addressed in this research, in particular, dispatching SCs to transport containers unloaded by QCs to the target locations in the yard. This is because the productivity of container terminals is often measured in terms of productivity of QCs [50], which are the most important and expensive equipments in port terminals. Efficiently dispatch the available SCs may greatly increase QCs productivity which directly improve the container terminal productivity.

2.4.2.1 Problem Description

For simplicity, only the container unloading process is considered in this research. The abstract model of the SC dispatching problem during the container discharging operation is shown in Figure 2.6 and described as follows.

To begin with, a number of QCs are assigned to discharge the berthed container vessel. When QCs start the unloading process, a number of SCs (N_s) queue up in the "SC waiting queue at transfer area" and wait to transport unloaded containers to the yard. Then two problems may occur during the discharging process:

- If N_s is too small i.e. insufficient number of SCs deployed, then one

or more QCs may have long idle times because each QC has a limited storage buffer below it. This problem is referred to as *crane starvation* problem in [31]. As the productivity of QCs may be measured by the time required to discharge and load container vessels, long idle time will greatly decrease QCs productivity which may in turn affect the container terminal productivity.

- If N_s is too large i.e. an excessive number of SCs deployed, then the SCs may have long idle times because they have to wait in the queue at the transfer point. This problem is called *blocking of the SCs before service* in [31]. Too many SCs running on the road might cause road blocking as well.

Therefore, the goal is to find the appropriate value for N_s in order to minimize the idle times of QCs and SCs.

2.4.2.2 Related Work

A number of prior research projects focusing on the vehicle allocation problem are summarized next.

Böse et al. [30] present methods of efficiently dispatching SCs to QCs assigned to a vessel in order to maximizing the QC productivity. Instead of static binding (i.e. fixed assignment) of SCs to each QC, a dynamic assignment strategy, where a predefined number of SCs serve several cranes, is suggested to increase the productivity of used SCs. Moreover, the dynamic assignment approach does not employ an optimization method. The further enhancement of the SC assignments is developed by applying a genetic algorithm. Several experiment results show that a genetic algorithm can be applied to minimize the ship berthing time.

Grunow et al. [51] present a simulation study of Automatic Guided Vehicle (AGV) dispatching strategies in a container terminal which takes into account stochastic parameters such as quay and stacking cranes handling times and release times of transportation orders. AGVs modeled in the simulation are capable of carrying one TEU container in single mode whereas they can carry one 2TEU container or two 1TEU containers in dual-carrier mode. A scalable simulation model is used to compare the pattern-based off-line heuristic proposed by the authors with a typical on-line dispatching strategy adopted from flexible manufacturing systems. The experiments results show that the proposed pattern-based off-line heuristic outperforms the traditional dispatching strategies.

In order to maximize the productivity of QCs, a queuing network model is proposed by Pietro et al. [31] for the dynamic management of a pool of QCs

and SCs each travels at the pre-defined speed. The proposed model is applied to analyze and evaluate the impact (e.g. crane's throughput and completion time) of alternative policies issued by the ship operation manager for all QCs. The claim is [31, p.5]: "the initial assignment of a fixed number of SCs to the same crane could be revised during operations in order to pursue the goal of balancing the unfinished work by each crane." The two problems stated and tackled in the paper refer to crane starvation and blocking of the SCs before service. The crane starvation (i.e. crane idle) problem occurs when an insufficient number of SCs are assigned to serve a crane. The second problem is due to an excessive number of SCs being assigned to serve a crane. An event graph that defines a set of events, system states, conditions and actions is proposed to solve the model.

2.4.2.3 Vehicle Dispatching Strategies

According to Meersmans and Wagelmans [76], vehicle dispatching strategies are very attractive especially in highly dynamic environments within which not all information is available or certain. container terminal is such a highly dynamic environment as discussed in Section 2.4.1.2. Thus, vehicle dispatching strategies may be appropriate for efficiently dispatching vehicles in container terminals. In this section is summarized the previous work related to vehicle dispatching strategies employed in various dynamic environments before three SC dispatching strategies are proposed.

Description

Vehicle dispatching strategies are popular in manufacturing systems where AGVs are often used to transport material between different locations [75]. Moreover, vehicle dispatching strategies do not use complicated mathematical models to work out the "best" vehicle assignment, and hence they are easy to implement [76]. Therefore, they are widely used in practice.

Egbelu and Tanchoco [40] propose several AGVs dispatching rules in a job shop manufacturing environment where a unit load (material) is handled by several work centres. Depending on the different points of view, vehicle dispatching strategies are divided into two categories, namely work centre initiated dispatching strategies and vehicle initiated dispatching strategies.

Under work centre initiated dispatching strategies, a work centre selects an idle vehicle to pick up a load. Whereas, under vehicle initiated dispatching strategies, the released vehicle selects a work centre to pick up its next load. The proposed dispatching strategies are summarized as follows:

- Work centre initiated dispatching strategies

  - Random Vehicle (RV): the work centre randomly assigns any available vehicle in the shop to the pickup task without regard to the location of the vehicle and the load.
  - Nearest Vehicle (NV): the work centre requesting service of any vehicle selects the available vehicle with the shortest travel distance or time.
  - Farthest Vehicle (FV): the work centre requesting service of any vehicle selects the available vehicle with the longest travel distance or time.
  - Longest Idle Vehicle (LIV): the vehicle with the longest idle time is selected by a work centre from a set of available AGVs to transport a load.
  - Least Utilized Vehicle (LUV): the work centre assigns the highest priority to the vehicle with the least mean utilization time.

- Vehicle initiated dispatching strategies

  - Random work centre (RW): the released vehicle randomly selects a work centre from a set of work centres requesting the service of vehicles.
  - Shortest Travel Time/Distance (STT/D): the released vehicle assigns the highest priority to the closest work centre.
  - Longest Travel Time/Distance (LTT/D): the released vehicle selects the farthest work centre to pick up its next load.
  - Maximum Outgoing Queue Size (MOQS): the released vehicle assigns the highest priority to the work centre that has the largest number of loads awaiting transport.
  - Minimum Remaining Outgoing Queue Space (MROQS): the released vehicle is dispatched to the work centre that has the smallest remaining space in its outgoing queue.
  - Modified First Come-First Serve (MFCFS): the released vehicle prioritizes work centres in chronological order by the transport service requesting times received from work centres.

Many researchers (e.g. René et al. [41], Le-Anh and René [43], Klein and Kim [44], Jeong and Randhawa [45], Bozer and Yen [46], and Srinivasan et al. [42]) have proposed the use of vehicle dispatching strategies in various highly dynamic environment such as warehouses, distribution centres, production plants, and container terminals.

Proposed Dispatching Strategies

The vehicle dispatching strategies are important issues for the performance of terminal operation [75], as using different dispatching strategies may result in different container terminal performance. Although vehicle dispatching strategies are usually used to dispatch AGVs, they can be used in the same way to dispatch manned vehicles such as SCs [41]. In this research, vehicle dispatching strategies are used to dispatch the released SCs to transport unloaded containers.

Three vehicle initiated dispatching strategies (i.e. a QC is selected by a released SC from a set of QCs having one or more unloaded containers awaiting transport) are proposed here and described as follows:

- Random (i.e. the RW rule described above). The released SC will randomly prioritize all working QCs that currently have one or more containers awaiting transport. If two or more containers are waiting to be moved, the one that has been waiting longest will be moved i.e. First-In-First-Out (FIFO). The FIFO rule attempts to minimize the waiting time of unloaded containers that are to be transported by SCs from the wharf to the yard.
- LargestJobNumber. When the released SC requires a job at the transfer point, the QC that has the largest number of unloaded containers will be selected by the SC. This is the same as the MOQS rule proposed by Egbelu and Tanchoco [40]. Additionally, when two or more QCs have the same number of containers to be transported, the QC that has the largest number of unfinished jobs (i.e. containers are yet to be unloaded from the vessel) will be assigned to the SC. The FIFO rule is applied if there are two or more containers awaiting transport under the QC.
- LongestTravelTime. When the released SC is looking for a job at the transfer area, the unloaded container that has the longest SC travel time (if this is not available for some reason then the average travel time from the wharf to the target yard block will be used) will be selected by the SC as its next job.

2.4.3 Use of Multi-Agent Systems for Evaluating Dispatching Strategies

Regarding the applicability of MASs in general, Parunak [32] discusses the usage of agent technology in industry from both researcher and industrial practitioner perspectives. In particular, he lists the following characteristics of problems to which agent technologies are considered as the most appropriate solution:

- *Modular*, that is an industrial entity has a well-defined set of state variables that are not coupled to those of its environment, and its interface to the environment can be clearly identified. As the agent naturally characterizes such an entity, the redeployment of the entity to the agent source code only requires minimal changes.

- *Decentralized*, agent *autonomy* and *pro-activeness* make agent technologies ideally suited to decentralized problems i.e. the system consists of distributed processes. Such a system can be decomposed into stand-alone software processes that are able to perform goal-directed behavior(s)/action(s) without direct intervention of other processes or external actors.

- *Changeable*, as some agent characteristics make them appropriate for the problems that are modular and decentralized, the combination of these characteristics makes agent technologies well suited to the application where frequent changes are anticipated. That is, agent technologies enable a system to change quickly, frequently, and with no side effects on the rest of the system.

- *Ill-structured*, agents are suited to the problem where not all information is available during the early phase of the system design. Agents are well suited to this type of problem because agents are designed to interact with the environment within which they are situated.

- *Complex*, agent technologies should be adopted for systems consisting of a large number of different interacting elements each of which interacts with each other in sophisticated ways.

Larry et al. [34] describe container terminal management as a decentralized, changeable, poorly structured, and complex problem. Its characteristics discussed in Section 2.4.1.2 further confirm that the container terminal management fits Parunak's characterization rather well. This would suggest that agent technology is indeed a viable solution to the container terminal management problem. Furthermore, a number of research projects have shown that agent-based technology is a promising approach to the problem and are summarized next.

Paul et al. [5] provide a comprehensive survey of previous research on agent-based technology that has been applied to the field of freight transport logistics. The survey shows that agent-based approaches (e.g. market-based approach) could be used to successfully solve many distributed and complex problems within transport logistics, especially for container terminals but that there is

a lack of verified deployed systems. Barbucha and Jędrzejowicz [6] propose an agent-based approach to solve vehicle routing problems. Computational experiments are conducted and the results show that the proposed agent-based approach produces good solutions to the targeted problems. Graudina and Grundspenkis [47] present an overview of the use of agent-based technologies to solve problems in the transport and logistics domains. They conclude that agent technology and open system architecture have been successfully applied in real life to solve problems within the transportation and logistics domains. Henesey [52] introduces a container terminal simulator based on a MAS approach to assist decision makers in evaluating container terminal management policies related to berth allocation for incoming ships and yard stacking. Their experimental results reveal that simulation can be useful for evaluating alternative management policies. Degano and Pellegrino [53] propose a real-time control system using Petri nets to monitor, diagnose and recover from unpredictable events in an intermodal terminal that is represented as a discrete event system.

Rebollo et al. [7, 8] propose a MAS approach to solve the problem regarding the automatic allocation of containers in a real port container terminal. In the proposed approach, the overall goal, the automatic container allocation, is decomposed into several sub-goals, each of which is achieved by a corresponding agent. A *Ship Agent* is created to minimize the loading/unloading time of a given ship and maximize the utilization of the employed QCs. The goal of a *Stevedore Agent* is to minimize the unnecessary moves of carriers (e.g. trucks, straddle carries, etc) assigned to a given QC. The goal of a *Service Agent* is to best allocate the arriving containers for a given stacking area. A *Transtainer Agent*'s goal is to maximize the utilization of an assigned transtainer and minimize its unnecessary moves. Finally, a *Gate Agent* is designed to control the arrival/departure of containers by land.

Specific to the research described in this thesis, a prototype MAS as an outcome of the research has been developed to evaluate and compare the performance of the proposed SC dispatching strategies during the container unloading operation. Similar to the approach proposed by Rebollo et al described above, a distributed approach has been taken to model different decision makers (e.g. ship controller, SCs) involved in the container discharging operation. The architecture of the prototype MAS is described in Chapter 3.

A MDA based approach is explored in this research to facilitate the development of the prototype MAS. The MDA introduced in the next section improves source code quality and reusability by raising the software design level from implementation to the abstract design model.

2.5 Model Driven Architecture for Multi-Agent System Development

The MDA [18], initially proposed by the Object Management Group in 2001, is a model-driven approach to designing and developing portable, interoperable, reusable software components and data models. It facilitates the development of complex software systems by automating the engineering process. In contrast to the conventional code-centric software development, the MDA based software development uses models as the primary engineering artifacts. Thus, it is necessary to have a clear understanding of what is a model in the MDA. For this purpose, the definition of a model within the MDA framework given by Kleppe et al. [22, p.16] is adopted here:

> **"A model is a description of (part of) a system written in a well-defined language.**
>
> **A well-defined language is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer."**

Therefore, a model is a specification of (part of) a system and is expressed in some particular language that is understandable by a computer.

There are currently two types of MDA based approach i.e. elaboration vs. translation. The elaborative MDA uses transformation tools to transform models at different levels of abstraction, elaborating some by inserting source code directly. On the other hand, in the translation approach (also called Executable UML [38]), the complete behavior of a system is specified in the high-level abstraction model, which is fully automatically translated into the executable code by an executable UML compiler. More specifically, the system's dynamic behavior is specified in the abstract model using a vendor-specific action semantics language. The elaboration approach is applied in this research to develop the prototype MAS due to the lack of a standardized syntax for the action semantics language and the low-level abstraction of the action language.

The adopted elaborative MDA approach explicitly separates the development process into models at three different levels of abstraction: *Platform-Independent Models* (PIMs), *Platform-Specific Models* (PSMs) and implementation models. A PIM is the high-level abstraction model of a system that is created independently of any particular implementation technology. In contrast, a PSM is a concrete model that is constructed by adding to the PIM the details of how an underlying implementation technology is to be used in the system implementation. An implementation model consists of all the information needed to construct a running system. The implementation model,

which is at the lowest level of abstraction, is automatically generated from one or more PSMs. The MDA defines a set of consecutive transformations that are automatically executed by tools to move from the high-level abstraction model to the implementation model.

2.5.1 Model Driven Architecture vs. Traditional Software Development

Great progress has been made since the early days of software development. This is evident in the growth of the complexity and size of the software systems that can be built nowadays. Nevertheless, software development is still immature and developers using the traditional code-centric approach face a number of problems. The MDA is a new software development paradigm that is able to address these problems [22]:

The Productivity Problem

The traditional software development is an effort-consuming process, which typically consists of the following six phases: *requirements gathering*, *analysis*, *design*, *implementation*, *testing*, and *deployment*. The system specifications and design models are created during the early phases. However, these high-level specifications and models become inconsistent with the real state of source code as soon as the implementation phase of the development process starts. This is because a majority of developers often consider updating the high-level system specifications and models as an overhead task. As the implementation phase progresses, the abstract system specification and models become outdated texts and diagrams. Hence, producing high-level specifications and models of a system in the first phases of the traditional software development is not being productive.

Using a MDA based approach may significantly reduce the software development time cycle, and thus costs. This is because model to model transformations and model to code transformations are all done automatically using a set of tools. The mappings from model to model and model to code, which are used to direct transformations, have to be defined by specialists who have knowledge about the platforms. Although defining such mappings is a difficult task, it only needs to be done once and can then be applied multiple times to productively develop a number of different systems.

The Portability Problem

As new technologies (e.g. Web Services, .NET, JEE, etc.) come to the market and become popular, the existing systems utilizing the older technologies

may be desired to be ported to the new technologies. This is because the new technologies offer tangible benefits that software companies can not afford to lag behind. As a result of applying the traditional code-centric approach, an existing system may need to be rebuilt each time a new technology is targeted. As a consequence, the investments in previous technologies lose value and they may even become worthless.

The MDA can effectively address this problem by separating the high-level abstraction model of a system from details of the system implementation. As the system's abstract model (PIM) is developed independently of any particular implementation technology, it is thus said to be portable. The same PIM can be transformed into multiple PSMs targeting different technology platforms. As a consequence, when new technologies come out in the future, the existing PIM can still be reused to construct the PSMs targeting the new technologies.

The Interoperability Problem

Today, software systems often comprise multiple components which are developed using different technologies, thus the interoperability that enables the distinct components to interact despite differences in programming languages and execution platforms is becoming a central issue. Adopting the traditional approach, developers have to spend much time and effort creating adapters to realize the interoperability between distinct components.

The MDA addresses this problem by using transformation tools to generate from the PIM not only the PSMs, but the necessary interaction bridges between them as well. The generated bridges are responsible for transforming and mapping the concepts from one implementation platform into the concepts used in another platform. As a consequence, software companies applying the MDA approach can cope with technology changes while preserving their investments in the PIM.

The Maintenance Problem

The traditional software development process is often driven by low-level coding, thus requirement changes are made directly in the system implementation and are not reflected in the high-level abstraction models. The inconsistency between the system's abstract models and implementation makes system maintenance much harder. Furthermore, maintaining the high-level abstraction models in sync with the implementation requires a lot of expensive resources.

The MDA approach allows developers to focus on the PIM, which has a higher level of abstraction than the implementation. Using a set of transformation tools, the PIM is transformed into the PSMs, which are in turn transformed

into the implementation. Thus, the PIM serves as the system high-level specification, which is consistent with the generated implementation. Moreover, requirement changes made to the system are realized by changing the PIM and regenerating the PSMs and the implementation. The guaranteed consistency between the system high-level abstraction model and the implementation minimizes the future system maintenance effort. Furthermore, the system implementation automatically generated from the PSMs is less error-prone and copes with changes in design more easily than with those generated manually.

2.5.2 Development Tools for Model Driven Architecture

A number of UML/MDA based CASE tools have been developed to ease the system modeling process and to automate or partially automate the transformations between the models. Some of the available CASE tools are summarized next.

QiQu [21] is an open source framework to support the MDA approach. Using QiQu, software designers can focus on modeling the core business functionality and generate the corresponding implementation in an automatic manner. QiQu takes a UML model represented in XML Metadata Interchange (XMI [84]). XMI is an OMG standard for interchanging UML models via XML documents. format as input and transforms the input model into an XML output model which is in turn transformed into source code (e.g. JAVA, C#, etc.). The first transformation is done using QiQu scripting language to define customized transformation rules. The second transformation is done using a template-engine to merge the transformed output model with the template(s). QiQu scripting language can also be easily extended if a required functionality is not available.

StarUML [19] is an open source software modeling tool that supports UML to develop a UML/MDA platform running on a Win32 platform. In order to give maximum support for software development using the MDA, it provides many customization variables like UML profile, MDA code and so on. The tool can also be extended by means of plugins and any user should have no problems in doing so because a reasonably comprehensive developer guide is available. A UML2XMI plug-in is available and can be used to export a UML model to an XMI document . The current version of the plug-in supports UML 1.3 [83] and XMI 1.1 [84]. In addition, an AML Profile for StarUML has been developed and is available on the Internet.

AndroMDA [20] is an open source, extendable code generation tool that adheres to the MDA specification. It is available on the Internet and free to download. As a transformation engine, AndroMDA takes UML models as in-

put and outputs deployable components for any underlying platform. It uses a plugin mechanism, the so-called *cartridges*, to generate source code for arbitrary target platforms or programming languages such as Java, C#, and so on. To build a tailored code generator, users can define their own cartridges or customize existing ones using a toolkit called meta cartridge.

The Eclipse Modeling Framework project contains a powerful source code generation tool called Java Emitter Templates [85]. Such a tool is an important component of Model Driven Development and plays an important role in minimizing redundant programming, increasing developer's productivity and facilitating application maintenance. Java Emitter Templates uses templates to generate XML, Java source code, etc. It takes a XML document as the input model and transforms the input into the code. Since the input model is an XML document, the XPath language [86] is used to query and navigate the model.

Chapter 3

Prototype Multi-Agent System Architecture

The overall system objective is to evaluate and compare the performance of the given set of vehicle dispatching strategies described in Section 2.4.2.3 during the container discharging operation. Thus, the best performed dispatching strategy can then be applied in practice to improve productivity of the container terminal.

A number of diverse and independent entities whose individual decisions may directly affect the performance of the others are involved in the container discharging process. Considering the distributed nature of the problem, an approach based on the multi-agent paradigm has been taken to design the system architecture. The MAS based approach divides the overall system objective into a number of independent objectives each of which is fulfilled by a type of agent. As shown in Figure 3.1, the prototype MAS comprises:

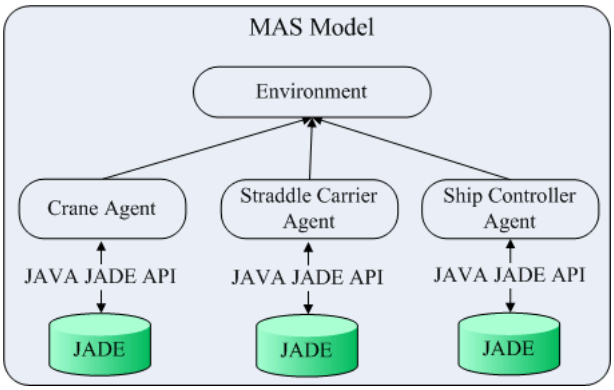

Figure 3.1: The prototype MAS Model.

- **Business Agents:** Three types of BDI structured agents are developed to model the independent decision makers involved in the container unloading processes. The ship controller is modeled as the Ship Controller Agent (CA) that is responsible for assigning unloading tasks to each Quay Crane Agent (QCA) and deciding which dispatching strategy is to be employed. The physical resources that are being used in the port terminal such as Quay Cranes, Straddle Carriers are modeled as QCAs and Straddle Carrier Agents (SCAs) respectively. Each QCA is responsible for unloading/loading containers from/to a container vessel. A SCA serves a QCA by transporting containers that are unloaded from the vessel to the yard.
- **Environment:** The container terminal environment is modeled as a Java Class. An instance of the environment class is created for each voyage. A voyage is a visit of a vessel to a terminal.
- **Persistence Mechanism:** Each agent has a capability to perform database related operations (updates and queries) on a local database. The Jade Software Corporation has developed a JADE-Java API that enables Java developers to utilize the JADE's OODBMS as a persistence mechanism and use the java object as a proxy. The prototype MAS will interoperate with JADE 6.2[1] through the JADE-Java API to store all information represented as objects that is needed to help the MAS achieve its goal.

The mental attributes of each type of agent presented in the architecture are specified in the following section.

3.1 Agent Description

3.1.1 Ship Controller Agent

The CA perceives its environment by reading the input voyage data that is retrieved from the Port Otago's JMT system using a JadeScript method (see Section 4.3 for more details). The obtained voyage data are then set to the agent's beliefs which can be classified into two categories: static or dynamic. Static beliefs represent information that can not be changed by the agent whereas dynamic belief represent information that may be changed by the agent during the MAS execution.

- Static beliefs
  - A predetermined ordered list of the containers to be unloaded from a container ship

[1] http://www.jadeworld.com/

  - A set of proposed SC dispatching strategies (see Section 2.4.2.3 for a discussion of these strategies) that are available for evaluation and comparison

- Dynamic beliefs
  - Total number of SCAs that serve all QCAs of one container vessel
  - A vehicle service queue.
  - The current vehicle dispatching strategy in use.

When the beliefs of CA are initialized, the agent operates accordingly to perform the following actions::

- Going through all of the proposed SC dispatching strategies for each voyage in order to compare their performance.
- Dispatching a list of job assignments to each corresponding QCA.
- Managing the queue of the idle SCAs. This includes registering and deregistering a SCA to the service queue.
- Sending a service notification to the first available SCA in the service queue.
- Updating its internal beliefs with respect to the perceived environmental changes.

The mental attributes of CA described above is depicted in Figure 3.2. In Figure 3.3 is shown the goal/plan structure of the CA. The *StarterPlan* plan is used to initialize the agent's local beliefs. Upon a new goal (*InitEnvironment*) is created, a dedicated plan (*InitEnvironmentPlan*) is selected and executed to perform environment initialization. After the environment is initialized, two new goals (*DispatchQCTask* and *DispatchSCTask*) will be created. The two goals trigger DispatchQCTaskPlan and DispatchSCTaskPlan respectively. The agent gets a request to register a SCA into the service queue by reception of a message (*msg: register_SCService*). A Plan (*RegisterSCServicePlan*) is responsible to process this message which leads to an update of the local beliefs. Similarly, a deregister service message (*msg: deregister_SCService*) is processed by an instance of *DeregisterSCServicePlan*.

3.1.2 Quay Crane Agent

In Figure 3.4 is described the mental attributes of a QCA. Each QCA has the following beliefs:

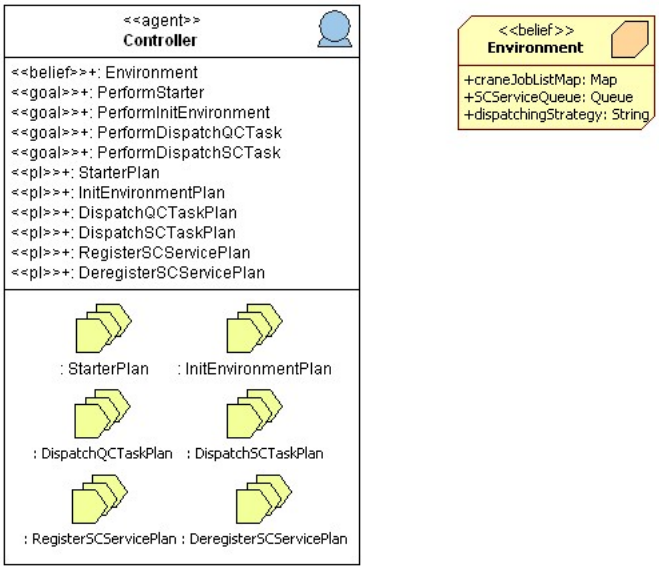

Figure 3.2: Mental attributes of the ship controller agent.

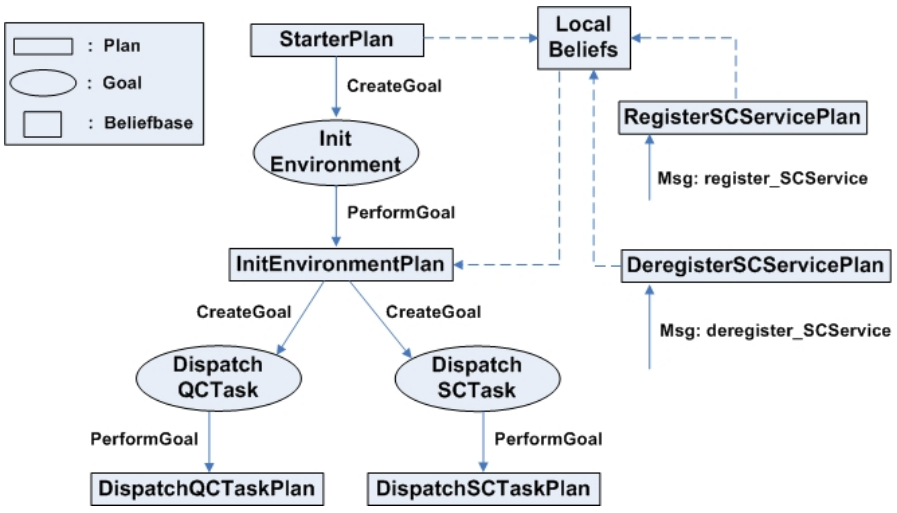

Figure 3.3: Goal/plan structure of the ship controller agent.

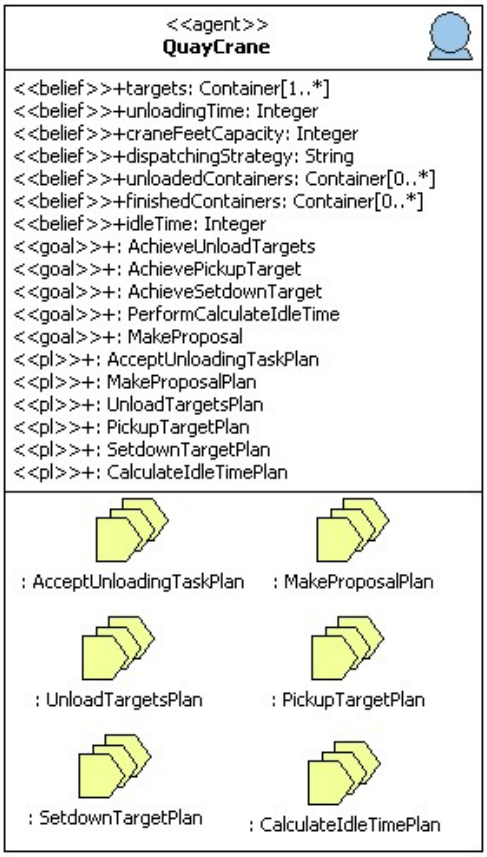

Figure 3.4: Mental attributes of a quay crane agent.

- Static beliefs
  - A list of unloading jobs (i.e. target containers to be discharged from a vessel) assigned by the CA
  - The discharging time of each unloading container
  - The maximum capacity of the storage buffer at the crane's feet area
  - The current SC dispatching strategy that is employed
- Dynamic beliefs
  - A list of unloaded containers awaiting in the storage buffer to be transported to the yard
  - A list of finished jobs i.e. unloaded containers which have been transported by a SC to the yard
  - Idle time of the QC due to overflows of the limited storage buffer below it

Four key goals for an individual QCA are identified:

- Unloading a list of containers from a vessel.
- Monitoring the storage buffer at its feet and pausing the discharging operation if the storage buffer reaches its maximum capacity.
- Performing calculation of idle time and accumulating it.
- Making a proposal when a carry request is received.

In Figure 3.5 is shown the goal/plan structure of a QCA. The *Msg: request(containers)* request message is processed by a plan (*AcceptUnloadingPlan*), which updates the agent's local beliefs. The plan will create a top level goal (*UnloadTargets*), which is achieved by a dedicated plan (*UnloadTargetsPlan*). There are two subgoals (*PickupTarget* and *SetdownTarget*) and one possible top level goal (*CalculateIdleTime*). The *PickupTarget* and *SetdownTarget* goals are achieved by instances of *PickupTargetPlan* and *SetdownTargetPlan* respectively. The *CalculateIdleTime* goal is created whenever the discharging operation is paused since the storage buffer reaches its maximum capacity. The calculation of idle time is performed by a plan *CalculateIdleTimePlane*. The call (*Msg: call for proposal*) is processed by a plan in each QCA (*MakeProposalPlan*).

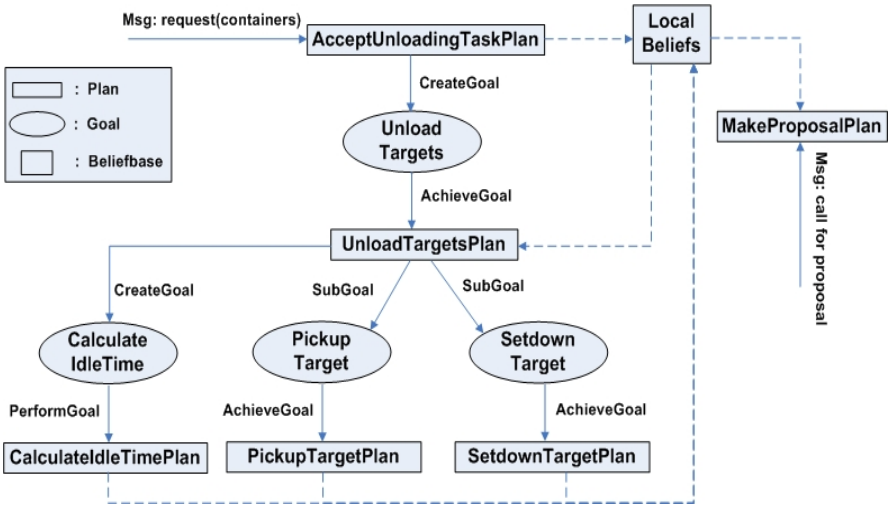

Figure 3.5: Goal/plan structure of a QCA.

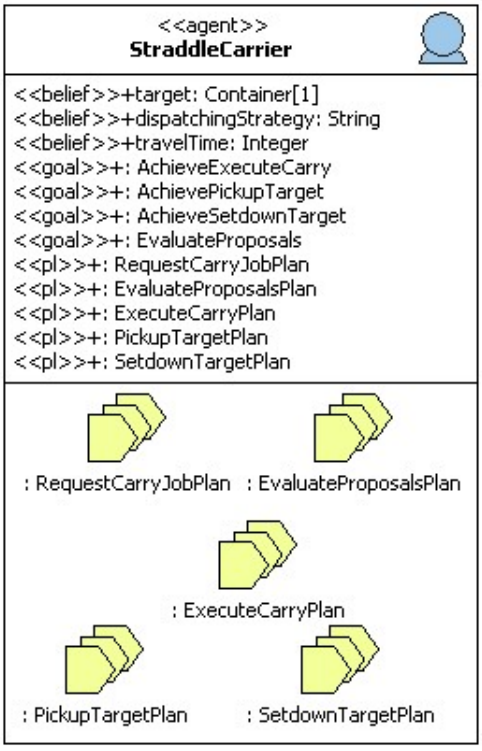

Figure 3.6: Mental attributes of a straddle carrier agent.

3.1.3 Straddle Carrier Agent

In Figure 3.6 is depicted the mental attributes of a SCA. The beliefs of an SCA are identified as follows:

- Static beliefs
  - The travel time of the carried container from the wharf to the yard
  - The SC dispatching strategy currently in use
- Dynamic beliefs
  - The next pick-up job

A SCA is modeled as an autonomous agent that has the following goals:

- Transporting a container from the wharf to the yard. This goal consists of two sub-goals i.e. pickup a container and setdown a container.
- Registering with CA when the SCA is ready for the next job.
- Deregistering with CA when the SCA is not available.
- Searching for the current working QCAs.
- Sending a ready for carrying notification to all of the working QCAs.
- Deciding which QCA should be served by comparing the received proposals.

In Figure 3.7 is shown the goal/plan structure of a SCA. The inform message *(Msg: inform_SCService)* is processed by a plan (*RequestCarryPlan*), which updates the agent's local beliefs. A plan (*EvaluateProposalPlan*) is used to evaluate proposals (*Msg: proposals*). Once accepted proponent has sent an inform message i.e. Msg: results(target) to specify the target container to be transported by the agent, the plan will create a new goal (*ExecuteCarry*). The goal is achieved by a dedicated plan (*ExecuteCarryPlan*). There are two subgoals (*PickupTarget* and *SetdownTarget*) which are achieved by instances of *PickupTargetPlan* and *PickupTargetPlan* respectively.

3.2 Agent Interaction

To illustrate the realization of the FIPA-RP and FIPA-CNP interaction protocols depicted in Section 2.3 on page 8 during the MAS execution, two particular scenarios are described as follows.

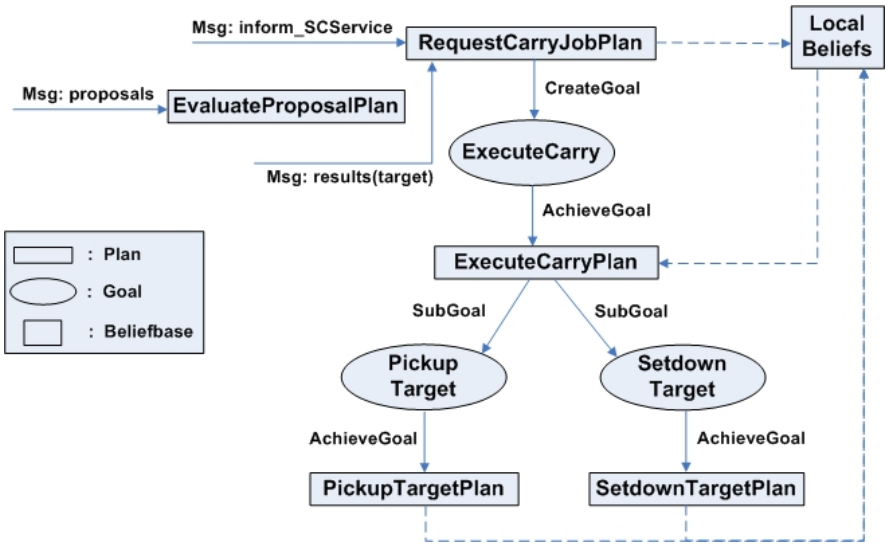

Figure 3.7: Goal/plan structure of a SCA.

In Figure 3.8 is depicted that a SCA requested the CA to perform registration action. The interaction between the two agents is realized according to the FIPA-RP interaction protocol. When a SCA returns to the transfer area after transporting a container to the yard, it tries to register with the CA. The SCA sends a request register message to the CA. If the CA successfully registers the SCA, it will send an inform message to the SCA otherwise it will send a failure to the SCA.

The LargestJobNumber strategy defined in Section 2.4.2.3 on page 19 has been employed to illustrate collaborations between agents for dispatching a SC to transport an unloaded container to the yard. In Figure 3.9 is shown a sequence diagram for SC dispatching collaboration that is realized based on the FIPA-CNP interaction protocol. To begin with, the released SCA (i.e. the first idle SCA in the service queue) requests a job at the transfer area by issuing a call for proposal (cfp) message to all the working QCAs. The QCAs receiving the cfp message will then be able to make proposals. Each proposal includes the number of unloaded containers awaiting transport and the number of unfinished jobs. Once the deadline has passed, the SCA evaluates the received proposals to select the QCA that has the largest number of containers to be transported. If all the proponents propose 0 unloaded containers awaiting transport then the SCA will reject all the proposals and reissue a cfp message. If two or more QCAs propose the same number of containers awaiting transport, then the proposal that has the largest number of unfinished jobs will be accepted. The rejected QCAs will be sent a reject-proposal message whereas the accepted proponent

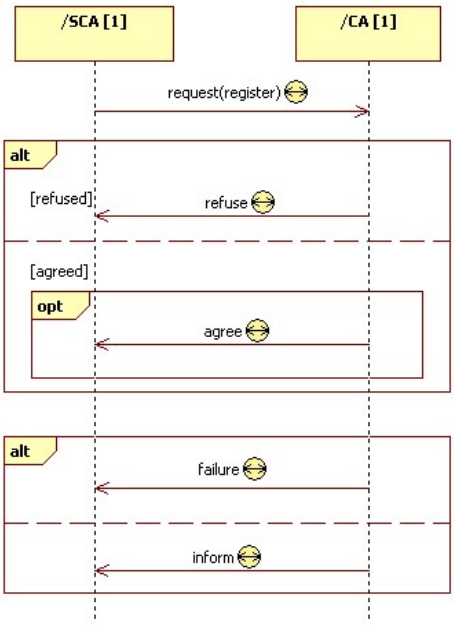

Figure 3.8: A sequence diagram depicts a SCA registering with the CA.

will receive an accept-proposal message. Once the selected QCA has prepared the target container to be transported, it sends an inform message to the SCA which will then transport the container from the selected QC to the target position in the yard. If an error occurs, a failure message will be sent back to the SCA which issued the cfp message. The FIPA-CNP interaction protocol is used to implement the other two proposed dispatching strategies (see Section 2.4.2.3) as well with the following variations:

- If the Random strategy is employed then the SCA will not evaluate the received proposals. Instead, it will randomly select a proposal which contains a positive number of containers awaiting transport. It will then send an accept-proposal message to the corresponding proponent and a reject-proposal message to the others.
- If the LongestTravelTime strategy is used then each QCA receiving the cfp message will propose the awaiting container that has the longest SC travel time. The SCA will evaluate the received proposals to select the container that has the LongestTravelTime and send an accept-proposal message to the corresponding proponent.

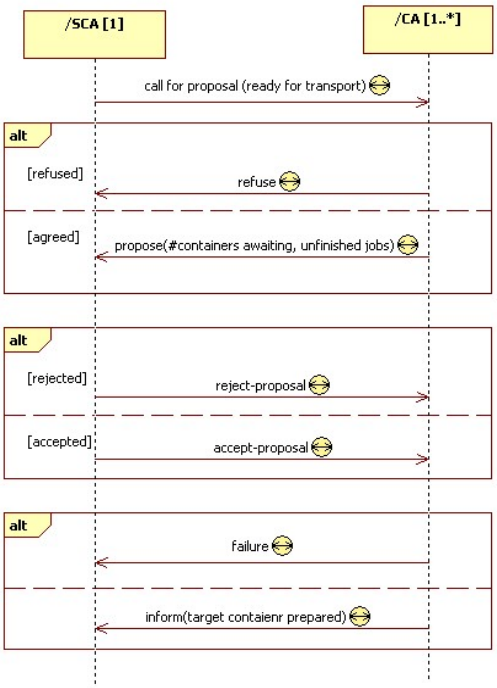

Figure 3.9: A sequence diagram for SC dispatching collaboration.

Chapter 4

Experiment

This section describes experiments conducted as a means of investigating the applicability of the MAS in the application domain of logistics. As applying different SC dispatching strategies may result in different performance of terminal operations, a MAS is used to assess the performance of the proposed SC dispatching strategies by running experiments. For a given vehicle dispatching strategy, the prototype MAS is applied to find the number of SCs that should be deployed for a container vessel during the container discharging operation, so as to minimize the idle times of working cranes, caused by overflows of the limited storage buffer below them. In addition, the main experimental objective is to have different number of SCs in each scenario (i.e. a voyage) to investigate the performance of the three proposed SC dispatching strategies during the container discharging operation. The performance criteria used is the number of SCs required to minimize QC delay time. A variety of real-world operation scenarios that vary in terms of different ship sizes and different numbers of QCs are used in order to correctly and comprehensively compare the effectiveness of the proposed dispatching strategies. Realistic input data of the experiment is taken from Port Otago's Jade Master Terminal (JMT) application [27], which delivers sophisticated ship planning and yard control that utilizes radio telemetry.

4.1 Experiment Setup

In the port of Otago, the current SC allocation is performed using one of the following strategies:

- Statically allocate a fixed number of SCs to one working QC.
- Allocate a fixed number of SCs to all the QCs that work on one container vessel.

Since each QC has a limited storage buffer at its feet area (three containers in the port of Otago), the first allocation strategy (i.e. static binding SCs to one QC) can be a bottleneck in some cases. For example, in the case of container discharging process, a QC has to pause the discharging operation if there are currently three unloaded containers waiting to be transported below the QC but none of the assigned SCs is able to move any of them in time. To avoid this situation as well as to increase the utilization of QCs and SCs, the second strategy (i.e. SC pooling) is employed in combination with the proposed dispatching strategies.

The productivity of a QC expressed using its working rate R can be formulated as the following:

$$R = \frac{n}{\sum_{i=1}^{n}(t_i + d_i)}$$

where

n being the total number of containers to be unloaded

t_i being the discharging time of the QC for the ith container

d_i being the QC's idle time (due to overflows of the limited storage buffer below it) when discharging the ith container

According to the above formula, if given the number of unloading containers n and the discharging time t for each container, then one possibility to enhance the QC productivity is to minimize the QC's total idle time i.e. $\sum_{i=1}^{n} d_i = 0$. Furthermore, as discussed in Section 2.4.2, the number of deployed SCs Ns have direct relation to the QC idle time d_i. That is an insufficient number of SCs may cause QCs to have long idle time, and an oversized number of SCs may cause SCs to have long idle time as well as congestion on the road. Therefore, it is necessary to find the appropriate value for Ns such that $\sum_{i=1}^{n} d_i = 0$.

Using an appropriate SC dispatching strategy will obtain a better value for Ns. Thus, the experiments go through all of the proposed SC dispatching strategies for each voyage in order to compare their performance. In each experiment run, the number of SCs is increased from low (i.e. 1) to high until the goal $\sum_{i=1}^{n} d_i = 0$ is achieved.

4.2 Experimental Assumptions

The following assumptions have been made and are kept the same for all dispatching scenarios:

ID	QCDischargingTime	SCTravelTime	SourceLocation	TargetLocation
String	Integer (sec.)	Integer (sec.)	String	String

Table 4.1: Properties of containers used in the experiments.

1. SCs operate continuously without any breakdowns.
2. A SC can carry only one TEU container at one time.
3. SC's loading/unloading time spent on the wharf and in the yard is constant. In the experiments, the stacking time of SCs is set to 0 for simplicity.
4. SCs always return to the queue at the transfer point after transporting a container to the yard.
5. SCs spend the same amount of time to transport a container from wharf to yard and return to the queue at the transfer area.

4.3 Experiment Input Data

The main performance criterion in this case study is minimizing the number of SCs to be deployed while ensuring all working QCs have no pause times. The input data for the experiment includes:

- Three different dispatching strategies: Random rule, LargestJobNumber rule and LongestTravelTime rule.
- 90 voyages recorded in 2007 are taken from Port Otago's JMT system which stores all the real-time information on container flows and various available resources. The input voyage data for the experiments, including number of SCs deployed, number of QCs used, containers to be unloaded by each QC are retrieved directly from the JMT system. In Table 4.1 is described the properties of containers used in the experiments.

As QCs are manually operated and the QC pickup/setdown time for each container unloaded from a vessel is not explicitly recorded, a container's discharging time has to be estimated empirically from the large set of historical data stored in the JMT system. A container's SC service time is non-deterministic because SCs are manually operated as well. Moreover, the SC service time is the sum of travel time, loading/unloading time spent on the wharf and in the yard. As the third assumption described in Section 4.2 states that SC's loading/unloading time spent on the wharf and in the yard is 0 for simplicity in the experiments, a container's SC service time is just its SC travel time. Although historical data documenting SC travel time of each container is available, some of them were

found unreliable (e.g. 2 sec. travel time - i.e. time not recorded correctly). In order to avoid using incorrect data, the average SC travel times between the wharf and different yard blocks are derived from the historical data recorded in 2006 and 2007. Therefore, when an incorrect SC travel time (i.e. travel time $\leqslant$ 10 sec.) is found in the input data of an experiment run, the corresponding average value will be used.

A JadeScript method is used to gather all the historical data needed for the experiments from the JMT system and save them into text files. JadeScript methods are written in the JadeScript class and can be executed from within the development environment (i.e. without the need to run a user application).

Extended Backus–Naur form (EBNF) is used to express the format of the input data file for the experiments. EBNF is an extension of the basic BNF metasyntax notation, which is a convenient means for expressing the grammar of a context-free language. EBNF is defined by the ISO-14977 standard [71]. The EBNF that is used to define the syntax of the experiments' input data files are presented as the following:

```
Voyage = {QC, jobList}, carriageReturn, SC;
QC = "CM_Crane", ",", identifier, ",";
jobList = {container};
container = "CM_Container", ",", identifier, ",",
            QCUnloadingTime, ",",SCTransportTime,",",
            sourceLocation, ",", targetLocation, ",";
SC = "CM_StraddleCarrier", ",", numSCDeployed, ",",
     numSCDeployed*identifier, ",";
numSCDeployed = digitWithoutZero;
identifier = alphabeticCharacter, {alphabeticCharacter | digit};
QCUnloadingTime = digitWithoutZero, {digit};
SCTransportTime = digitWithoutZero, {digit};
sourceLocation = "C", digitWithoutZero, "TA", whiteSpace,
                 digitWithoutZero;
targetLocation = alphabeticCharacter, whiteSpace, {digit},
                whiteSpace, digitWithoutZero;
alphabeticCharacter = "A" | "B" | "C" | "D" | "E" | "F" | "G"
                    | "H" | "I" | "J" | "K" | "L" | "M" | "N"
                    | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
                    | "V" | "W" | "X" | "Y" | "Z";
digit = "0" | digitWithoutZero;
digitWithoutZero = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
whiteSpace = ? US-ASCII character 32 ?;
```

```
carriageReturn = ? US-ASCII character 13 ?;
```

See Appendix B for an example of experiment input data file whose format is defined by the above EBNF.

4.4 Experimental Environment

A prototype MAS has been developed in order to investigate the performance of the three proposed SC dispatching strategies in various container unloading scenarios. The proposed MAS architecture, described in the last chapter, comprises three different types of BDI agents, including the Quay Crane Agents (QCAs), Straddle Carrier Agents (SCAs) and the Ship Controller Agent (CA). QCAs and SCAs represent physical resources of the system. Similarly, the CA models the ship controller in the port terminal. For each voyage scenario, a group of these agents (i.e. several QCAs, several SCAs, and one CA) try to accomplish the overall system goal by pursuing their own goals in coordination with other individuals.

4.5 Experiment Results and Analysis

This section presents the experiment results and corresponding evaluations. The results should be considered as preliminary due to the fact that a number of assumptions, presented in Section 4.2, have been made throughout the experiments. For example, SC technical failure is not considered in the experiment, however it may influence on the number of SC required in a real-world situation.

In Figures 4.1, 4.2, 4.3 and 4.4 are depicted the performance (measured using QC idle time in relation to the number of SCs) of the proposed SC dispatching strategies (defined in Section 2.4.2.3) for unloading 100, 201, 324 and 442 containers respectively. The number of containers to be unloaded was picked up randomly. The realization of these strategies during the prototype MAS execution is described in Section 3.2. Basically, all the four figures show that as the number of SCs increases, the QC idle time decreases. When the number of deployed SCs is quite low (e.g. one), the LargestJobNumber rule always performs worse than the random rule. As the number of SCs increases, in all cases, the QC idle time performance of the LargestJobNumber rule improves dramatically, the performance of the random rule and the LongestTravelTime rule are close to each other. All the four figures show that the LargestJobNumber rule has the best overall performance i.e. the smallest number of SCs required to achieve $\sum_{i=1}^{n} d_i = 0$ or $\sum_{i=1}^{n} d_i \leq \mu$ where μ being the acceptable QC idle time (e.g. 1000 sec.).

In Figure 4.5 is shown a summary of 90 voyages archived in 2007. It shows that as the number of unloading containers increases, the number of deployed SCs increases, but the number of QCs does not vary much. For the majority of voyages, two or three QCs are required. When the number of unloaded containers increases from 0 to 200, the number of deployed SCs have a positive trend with the peak values at 12 SCs required to serve two QCs for unloading 200 containers. Whereas, the number of deployed SCs does not vary significantly (i.e. the trend is about flat), when the number of unloaded containers increases from 200 to about 540.

In Figure 4.6 is given the summary of the average performance of the proposed SC dispatching rules for 90 voyages processed in 2007. The average of realistic data served as the baseline of the performance comparisons is shown in the figure as well. On average, two QCs are required. The experiment result suggests that the LargestJobNumber dispatching rule has the best overall performance, whereas the random rule has the worst performance. When the number of unloading containers is less than 30, the performance of all three dispatching rules are quite close to each other. For all the three dispatching rules, the trend of required number of SCs follows the trend of the baseline across all the scenarios.

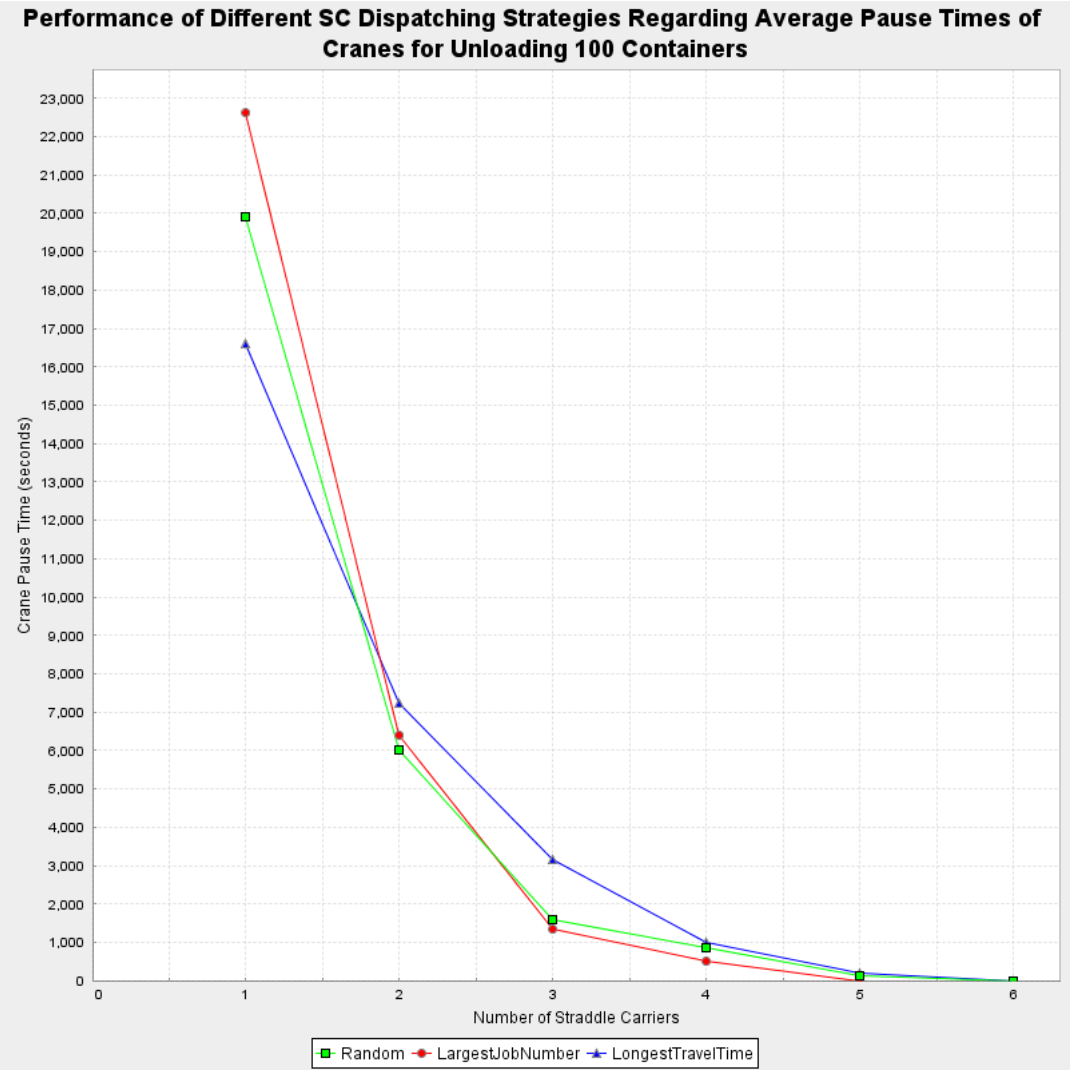

Figure 4.1: Crane idle times for different number of SCs and dispatching Rules.

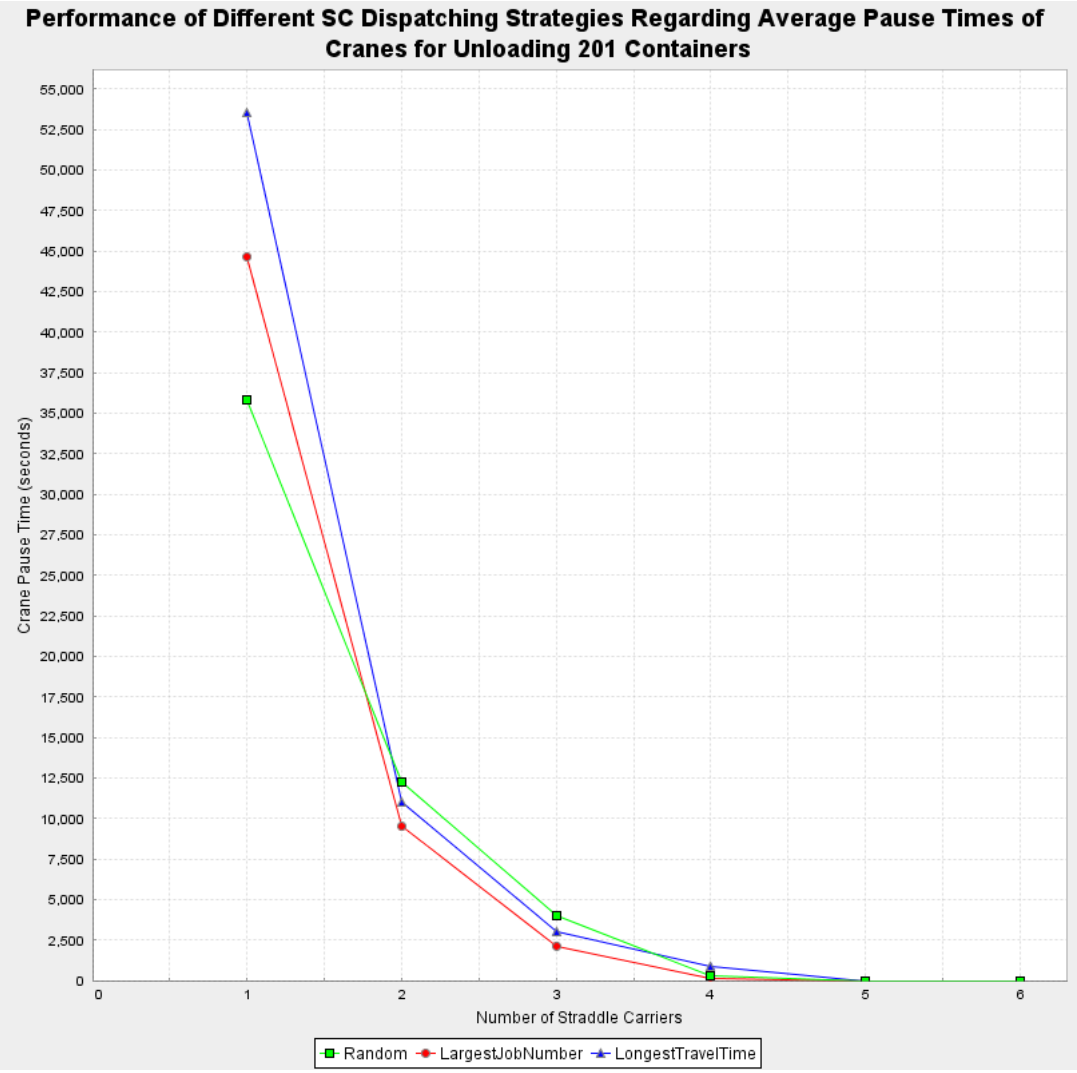

Figure 4.2: Crane idle times for different number of SCs and dispatching Rules.

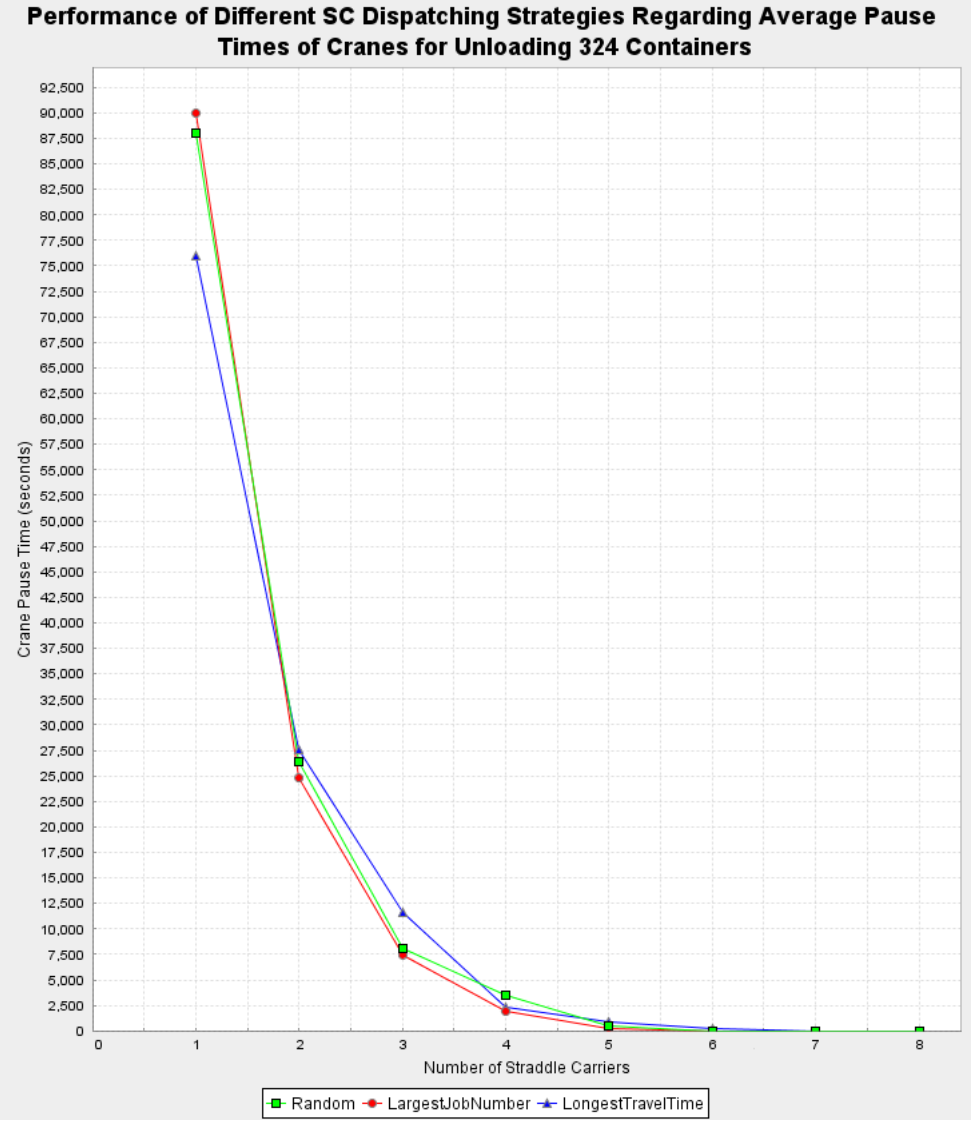

Figure 4.3: Crane idle times for different number of SCs and dispatching Rules.

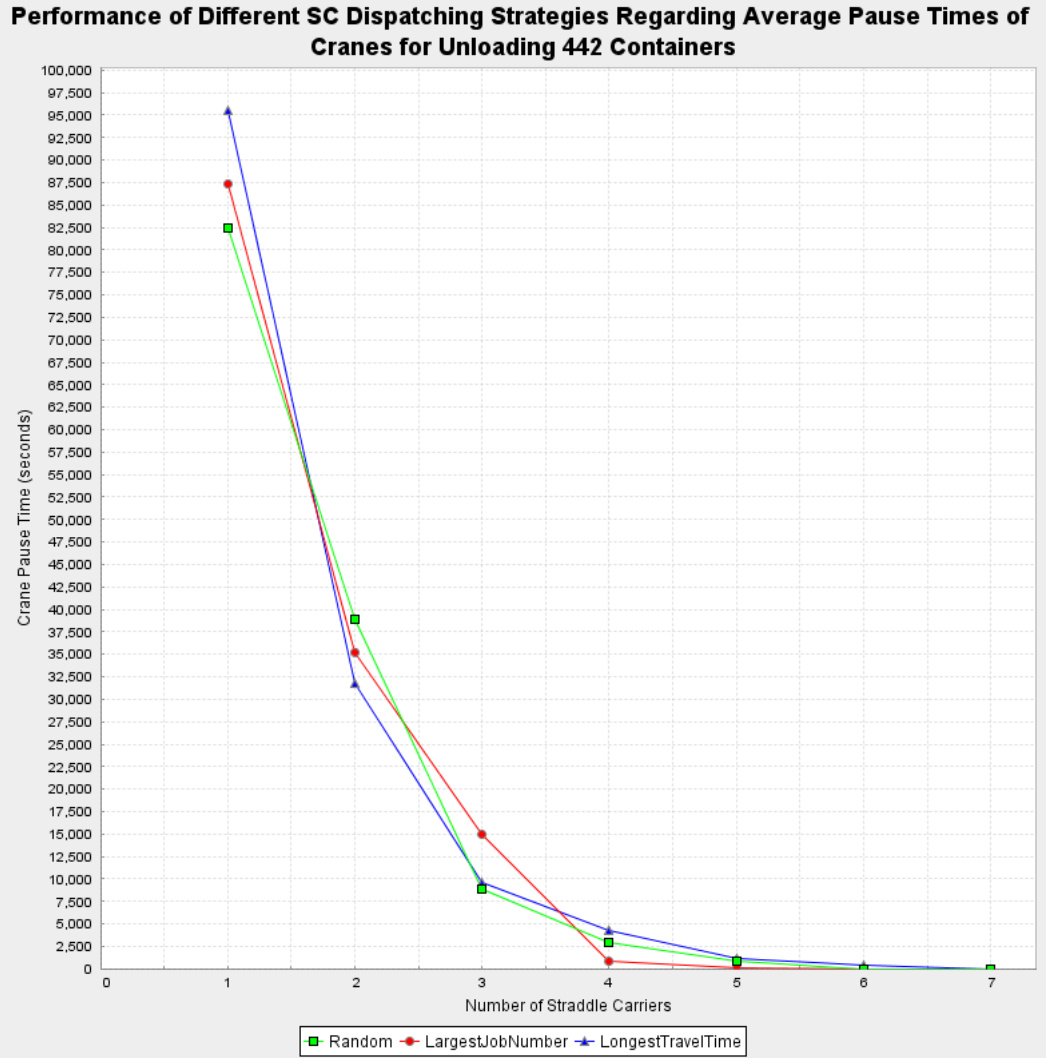

Figure 4.4: 'Crane idle times for different number of SCs and dispatching Rules.

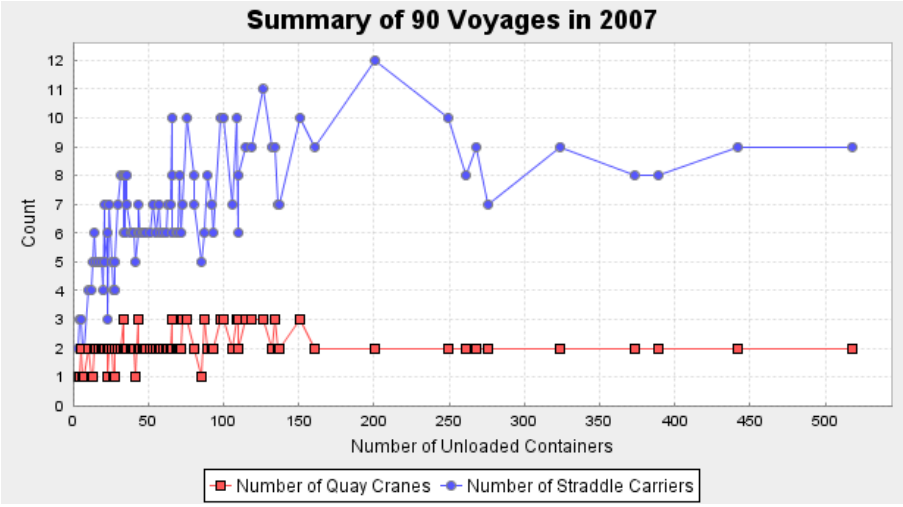

Figure 4.5: Summary of 90 Voyages in 2007

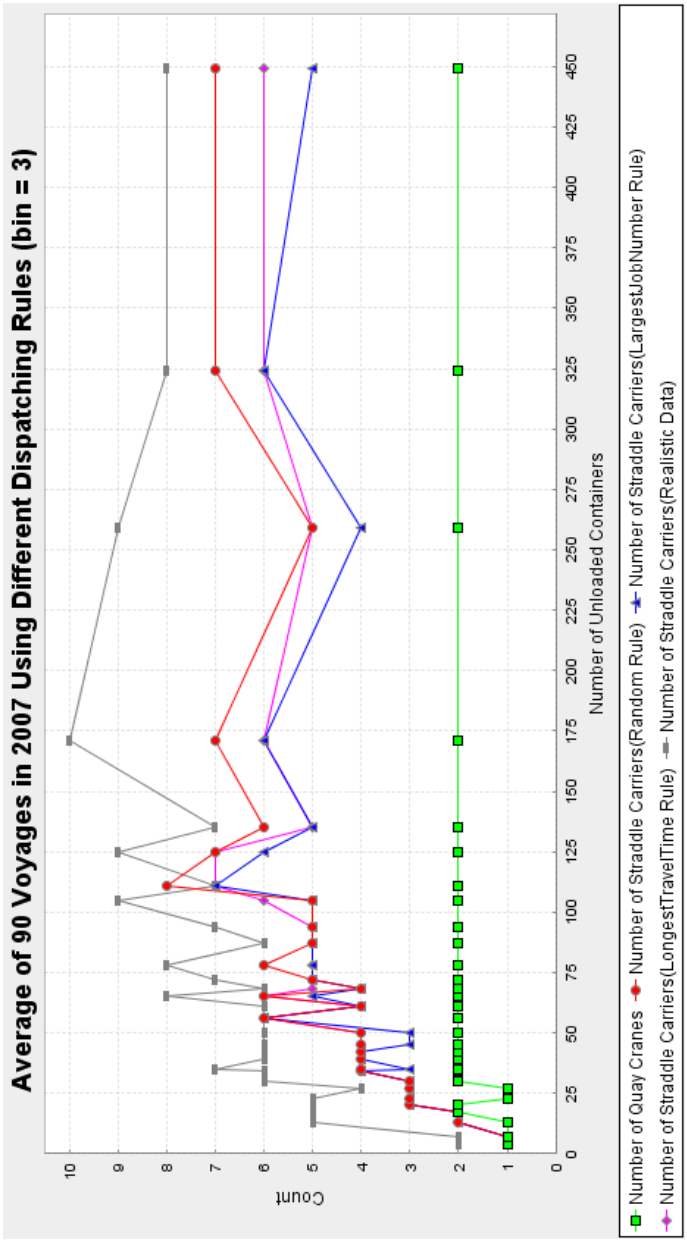

Figure 4.6: Performance of different dispatching rules for average of 90 voyages recorded in 2007.

Chapter 5

Model Driven Architecture for Developing the Prototype Multi-Agent System

In this chapter is given the author's answer to the research question 2 (see Section 1.2) how to apply the MDA to agent technologies, providing a partially automated support for the derivation of MAS implementation from the agent-oriented design, independently from the target implementation platforms. In order to realize the MDA process for the development of the prototype MAS (see Chapter 3 for more information about the MAS), the author has developed a high-level abstraction MAS model, as well as a set of transformations needed to transform the abstract model into two related platform specific models and further to generate the system implementation. More specifically, the MDA process described in Section 5.1 explicitly separates the MAS development into models at three different levels of abstraction: PIM, PSM and implementation. The MAS PIM, as presented in Section 5.2, is a high-level abstraction system model that focuses on the problem definition by describing the system in an agent-oriented level. The PIM that is independent of any particular implementation platform comprises two components, business logic and database, each of which is under a separate control. Thus, each component of the PIM can be transformed into a separate PSM targeting a different implementation platform. The Jadex BDI reasoning engine is selected for the development of goal-oriented agents; JADE, an object-oriented software development platform that includes an object oriented database management system, is employed as an object persistence mechanism to store all information that is needed to help the MAS achieve its goals. In Section 5.3 is described the transformations used

to transform the PIM into the Jadex PSM and the JADE PSM. In order to construct a running MAS, the two PSMs are finally transformed into the corresponding implementation. The PSM-to-Implementation transformations are presented in Section 5.4. As the two PSMs generated from the PIM target different implementation platforms, they cannot directly interact with each other. This has created a need for interoperability. In Section 5.5 is described how the MDA addresses this problem by generating from the PIM not only the two PSMs, but the necessary interaction bridge between them as well. The generated bridge is responsible for transforming and mapping the concepts from one implementation platform into the concepts used in another platform.

5.1 The Model Driven Architecture Process

In this section is presented the model-driven process, adopting a set of core standards of the MDA, for modeling and implementing the prototype MAS.

5.1.1 Adopted Model Driven Architecture Standards

Meta Object Facility (MOF)

MOF [88] is an OMG standard that defines the language to define modeling languages. All of the modeling languages, which are standardized by OMG and used in the MDA process, need to have formal definitions so that tools will be able to automatically transform the models written in those languages [22].

Unified Modeling Language (UML)

UML [9], as a foundation of MDA, provides a set of standard graphical notations that can greatly help in developing software systems by raising the level of abstraction from programming languages to models. The UML is an industry-standardized modeling language defined in the MOF for specifying, constructing and documenting software system models at various levels of abstraction.

UML Profile

UML profiles were introduced to attach additional semantics, properties and constraints to existing UML elements using stereotypes, tagged values and constraints respectively (for more detail, see Section 4.6 of [87]). The extended language can then be used to create UML models. Any number of profiles can be applied to an existing UML model. A list of standard UML profiles created for specific purposes are available from OMG's Profiles Catalog [11].

XML Metadat Interchange (XMI)

XMI [84] is a standard way, defined using the MOF, to generate an interchange format based on XML for models defined in a modeling language whose metamodel is described in the MOF [22]. XMI has mainly been used as an interchange format for UML models.

5.1.2 Selected Development Tools

Some of the available CASE tools described in Section 2.5.2 are selected to support the MDA process for developing the prototype MAS:

QiQu scripting language is employed as the mapping language for defining PIM-to-PSM transformation rules according to the applied UML profiles. QiQu-Velocity template engine is used to develop the PSM-to-Implementation transformation by merging the transformed PSMs represented in XMI with a set of predefined templates. QiQu has been chosen to develop model transformations because: (a) it is an open source framework to support the MDA; (b) QiQu provides a highly flexible scripting language, which heavily relies on XML, so that users can not only convert from XMI (representing UML models) to source code, but from any XML format into anything else; (c) it lets users build their own domain-specific generators that transform their models into the code of their choice. Hence, users are not limited to any predefined transformation engines or any predefined transformation rules.

StarUML is selected as the UML modeling tool where the UML profiles are applied to create the PIM and to refine the PSMs. The reasons for this tool choice are: (a) it is an open source project; (b) a UML2XMI plug-in (for exporting/importing UML models to/from XMI documents) is available; and (c) an AML profile (see Section 5.2.1) implementation that is specific to StarUML has been developed and is available on the Internet.

5.1.3 Process Stages

As depicted in Figure 1.1 on page 3, the MDA process for semi-automatically developing the MAS models at three different levels of abstraction is composed of the following three stages:

1. In the first stage, the MAS PIM, which is a UML model, is manually created by applying one or more appropriate UML profiles. The UML profiles, which are independent of any underlying agent implementation platform, for modeling systems in an agent-oriented level are particularly interesting. The Agent Modeling Language (AML) profile is such a UML profile that has been selected for creating the MAS PIM (see Section 5.2.1

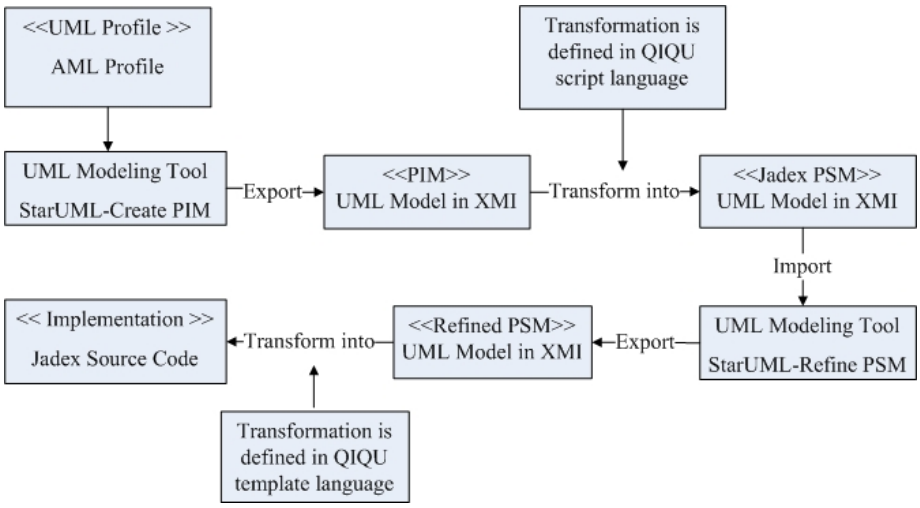

Figure 5.1: Model transformation process.

for the reasons of this choice). Once the PIM is created, it is then exported from StarUML to a XMI document.

2. In the second stage, a set of transformation rules are developed in order to transform the resulting MAS PIM in the format of XMI into two related PSMs represented in XMI: the Jadex PSM and the JADE PSM. In addition to the PSMs, an interaction bridge is generated to enable the interoperability between the two PSMs. As the generated PSMs need to be expressed in terms of their corresponding UML profiles, two platform-specific UML profiles are created by the author to respectively address agent and data persistence concerns at the implementation level of abstraction. The Jadex profile provides constructs for modeling Jadex agents. The JADE profile provides constructs for modeling data to be persisted in JADE. After the PSM generation process is completed, the PSMs, represented as an XMI document, can then be imported into StarUML for further refinement.

3. In the final stage of the MDA process, a collection of transformation rules are developed in order to transform the refined PSMs and the interaction bridge, which are exported from StarUML to an XMI document, into the corresponding implementation.

In Figure 5.1 is shown the model transformation process described above for semi-automatically transforming the MAS PIM into the Jadex PSM which is in turn transformed into the corresponding implementation.

5.2 Platform Independent Model

In this section is presented the development of the MAS PIM. A PIM describes the core business functions independently of any specific hardware, operating system, or programming language. In this research, the system PIM, which is described in Chapter 3 on page 27, is a high-level abstraction system model that focuses on the problem definition by describing the system in an agent-oriented level. The MAS PIM comprises two components, business logic and database, each of which is under a separate control. As the starting point of the MDA process, the MAS PIM is manually created by applying one or more UML profiles in StarUML.

5.2.1 Modeling Agents

Two UML profiles that may be used to model MASs are the AML profile and the AUML profile.

AUML [16, 17] is an agent-based unified modeling language that is being developed by the FIPA Modeling Technical Committee[1] for capturing the unique features of MASs (e.g. dynamic interactions between agents). It maximizes the reuse of standard UML. More specifically, AUML extends the standard UML by introducing new classes of diagrams to fulfill the distinctive requirements of MASs. For example, the AUML agent interaction protocol diagram is an extension to the UML sequence diagram for modeling interactions between agents.

AML [12] is a semi-formal visual modeling language developed by Whitestein Technologies[2] for specifying, modeling and documenting MASs. It was designed to improve the current MAS modeling languages and fill in all the missing aspects (e.g. insufficiently documented and non-intuitive modeling constructs). AML is also supported by CASE tools such as StarUML which is described in Section 2.5.2. Ivan and Radovan [14] discuss various aspects of AML in accordance with major OMG modeling frameworks (MDA, MOF, UML, and OCL [15]).

In this research, AML was chosen as the modeling language for developing the MAS PIM since it is:

1. A platform independent modeling language which does not restrict the implementation of its models to any specific implementation platform. As a consequence, the MAS PIM created using AML is a high-level abstraction model that is portable to different agent implementing platforms.

[1] http://www.fipa.org/activities/modeling.html
[2] http://www.whitestein.com/pages/company/about.html

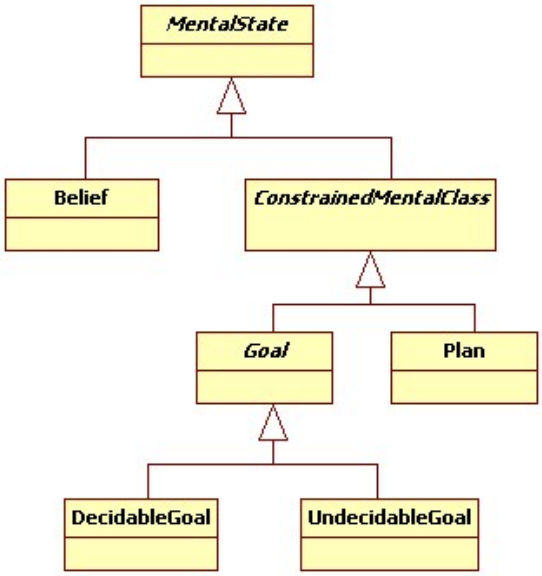

Figure 5.2: Metamodel for modeling mental attributes of BDI agents.

2. A modeling language that is MOF compliant. This characteristic facilitates the MAS development process since XMI can be used during the development process.

3. One of the most advanced and comprehensive languages available for specifying, modeling and documenting MASs [13][14].

4. Supported by CASE tools such as StarUML. An AML profile developed for StarUML is currently available.

The *Mental* package of AML defines the metaclasses for modeling autonomous agents' mental attributes (beliefs, goals and plans) that represent the agents' informational, motivational and deliberative states. These metaclasses are depicted in Figure 5.2. MentalStates referred to by several agents simultaneously represent their common mental states such as common beliefs and goals.

MentalState

MentalState is an abstract metaclass that is a common superclass to all metaclasses used for specifying mental attributes of BDI agents.

Belief

Stereotype: $\ll belief \gg$

Figure 5.3: An example of Belief.

Constraint	Stereotype	Semantics
preCondition	*pre*	The condition that must be true before the ConstrainedMentalClass can become effective (i.e. a goal can be committed to or a plan can be executed).
commitCondition	*commit*	The condition under which an agent could commit to the particular ConstrainedMentalClass instance.
invariant	*inv*	The condition that must hold for the ConstrainedMentalClass instance to remain effective.
cancelCondition	*cancel*	The condition under which an agent cancels attempting to accomplish the ConstrainedMentalClass instance.
postCondition	*post*	The condition that holds after the instance of ConstrainedMentalClass has been accomplished.

Table 5.1: Mental constraints for a goal.

Belief is a metaclass used to model beliefs of a BDI agent. An instance of Belief represents the information that the agent believes but which is not necessarily regarded to be objectively true. A Belief can have attributes and/or operations to represent its parameters and functions. In Figure 5.3 is shown a Belief (*OperationCompleted*) that represents an agent's informational state i.e. the current completion status of some operation. The default value of the attribute *isCompleted* shows the operation is not completed, but this value can be changed at runtime.

ConstrainedMentalClass

ConstrainedMentalClass is an abstract metaclass that extends MentalClass and allows its concrete subclasses to specify one or more mental constraints. In Table 5.1 is given a summary of the available mental constraints.

Goal

Goals represent objective environment states that the agent desires to

Figure 5.4: An example of DecidableGoal

achieve or maintain. An instance of a Goal represents the agent's commitment to the goal. Goals can have attributes and/or operations to specify parameters and utility functions of their instances. A Goal may specify one or more mental constraints used within the agent's reasoning processes. Goals are classified into two concrete types, namely *DecidableGoal* and *UndecidableGoal*.

DecidableGoal

Stereotype: $\ll dgoal \gg$

DecidableGoal is a specialized Goal used to model the type of goals that an agent can decide whether the goal has been achieved or not. In Figure 5.4 is presented a DecidableGoal (*AchievePickupTarget*) that represents an agent's motivational state i.e. pick up a target container. The goal has an attribute (*target:Container*) that will be initialized when the goal becomes the actively pursued goal. In addition, a postCondition (i.e. the currently carried container must be the target container) is specified for deciding whether the goal has been achieved.

UndecidableGoal

Stereotype: $\ll ugoal \gg$

UndecidableGoal is a specialized Goal used to model the type of goals that an agent can not decide whether the goal has been achieved or not. UndecidableGoal can be used to describe non-functional requirements of a system. This type of goals is not used in the prototype MAS, an example can be found in the AML specification [90] on page 155.

Plan

Stereotype: $\ll plan \gg$

Plan is used to model predefined plans of a rational agent. A Plan instance specifies a course of actions that the agent will perform in order to pursue the chosen goal. Like beliefs and goals, plans can have attributes and/or

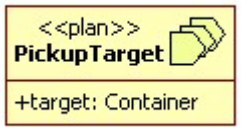

Figure 5.5: An example of Plan

operations to specify parameters and utility functions of their instances. Specific to the research described in this thesis, plan parameters are used to capture parameters from a goal that triggered the plan. Since plan is a specialized ConstrainedMentalClass, it can specify mental constraints.

In Figure 5.5 is shown a Plan (*PickupTarget*) that represents the means of pursuing the goal (i.e. pickup a target container). The plan has an attribute (*target:Container*) that specifies the goal parameter to be mapped. In addition, a preCondition (i.e. the target container is not null) is defined to the *PickupTarget* Plan.

5.2.2 Modeling Persistence Data

StarUML supports a number of UML standard elements (e.g. stereotypes and tagged values), among which the tagged value *{Persistence=PERSISTENT}* specified for a class denotes [83, p.2-28]: "the permanence of the state of the classifier". In developing the MAS PIM, this tagged value is used for marking classes whose instances are to be persisted in a database.

5.3 Platform Independent Model to Platform Specific Model Transformation

Model transformation, as described in Figure 5.6, is the process of converting a model (e.g. the PIM) into another model (e.g. a PSM) using a collection of mappings. A mapping is a transformation specification defined in some particular language, for instance QiQu scripting language. The PIM is created using a platform independent UML profile such as the AML profile described in the previous section. Using a set of mappings, the PIM can then be transformed into a PSM that is expressed using a second, platform specific UML profile. In this case, mappings specify transformation rules for automatically transforming the elements, such as stereotypes and tagged values, defined in the platform independent UML profile into the corresponding elements provided with the

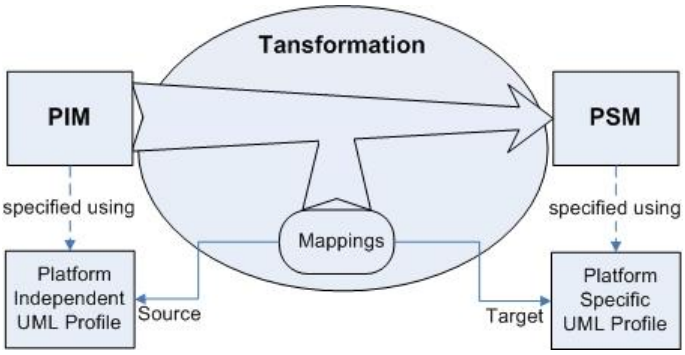

Figure 5.6: Model transformation.

platform specific UML profile. Furthermore, mappings are not available for those elements defined in the platform specific UML profile which are exclusively specific to the target PSM.

The model transformations from the MAS PIM to the Jadex and JADE PSMs, as discussed in Sections 5.3.1 and 5.3.2 respectively, represent the second stage of the MDA process described in Section 5.1.3. In order to utilize UML constructs to model Jadex and JADE concepts, the author has created two platform specific UML profiles. Following the creation of the Jadex and JADE UML profiles, the author has created two sets of mappings for transforming the MAS PIM prepared using the AML profile into the Jadex and JADE PSMs, expressed in terms of their corresponding UML profiles, respectively.

5.3.1 Platform Independent Model to Jadex Platform Specific Model

While there exist a number of realizations of the BDI model (e.g. [23], [59], [60] and [61]), this research has adopted the Jadex BDI reasoning engine (Jadex) for the development of goal-oriented agents. This is because Jadex fully supports BDI agents' practical reasoning process consisting of goal deliberation and means–end reasoning in contrast to all other available BDI engines which only support the means-end reasoning phase of the reasoning process (see Section 13.4 of [65]).

Jadex Description

Jadex uses beliefs, goals and plans to explicitly represent an agent's mental attitudes.

Beliefs represent an agent's information about itself and the world. Aiming for ease of use, Jadex has a simple representation of beliefs and thus does not support any (e.g. logic-based) inference mechanism. More specifically, two types

of beliefs are supported i.e. *belief* and *beliefset*. Each belief/beliefset has a name and a value called *fact*/*facts* which can be an arbitrary Java object. Beliefs can be used as conditions that trigger plans or goals and are monitored by Jadex for relevant changes.

Goals represent the concrete desires of an agent and thus drive the agent to perform appropriate actions to pursue them. Jadex defines a hierarchical structure of goals: top-level goals may have subgoals which are created from a plan.

Four types of goals are available: perform, achieve, query and maintain goals. A *perform goal* represents a desire of an agent to perform an action. This type of goal is considered to be achieved when the required action has been executed regardless of the outcome. In contrast, an *achieve goal* specifies a target future world state that an agent wishes to reach. An agent may try out alternative plans to achieve this type of goal. A *query goal* is related to an internal state of an agent and expresses a need of the agent for information. The outcome of this type of goal is the availability of certain information that the agent wants to know about. Finally, a *maintain goal* represents a desired world state that should be maintained by the agent under all circumstances. A violation of the maintained state will cause the agent to continuously execute appropriate plans to re-establish the desired world state.

To describe the situations in which a goal is instantiated, suspended, or dropped, the *creation, commit,* or *drop* condition can be applied. The *target* condition specifies the condition under which a goal can be considered as achieved; the *failure* condition is used to describe the opposite; Finally, the *maintain* condition is applied to depict a specific state that is desirable to be monitored and maintained.

Plans are the means of pursuing goals. A Jadex plan comprises two distinct components: the *plan head* and the *plan body*. The plan head specifies the conditions under which the plan will be selected and executed, e.g. receiving a message, activating a goal which the plan may handle. Conditions can be used to abort a running plan on demand. The plan body, which is implemented in a concrete Java class, defines a course of action to be executed when the plan is selected. The body class extends a base class of the existing Jadex framework in order to perform actions provided by the system API such as sending messages, manipulating beliefs or dispatching subgoals.

Instead of introducing a new agent programming language, Jadex employs existing well-established technologies such as XML and Java for programming agents. As illustrated in Figure 5.7, a Jadex agent's static structure comprising beliefs, goals and plan heads is specified in an Agent Definition File, which is an XML file that conforms to the Jadex meta-model represented as an XML

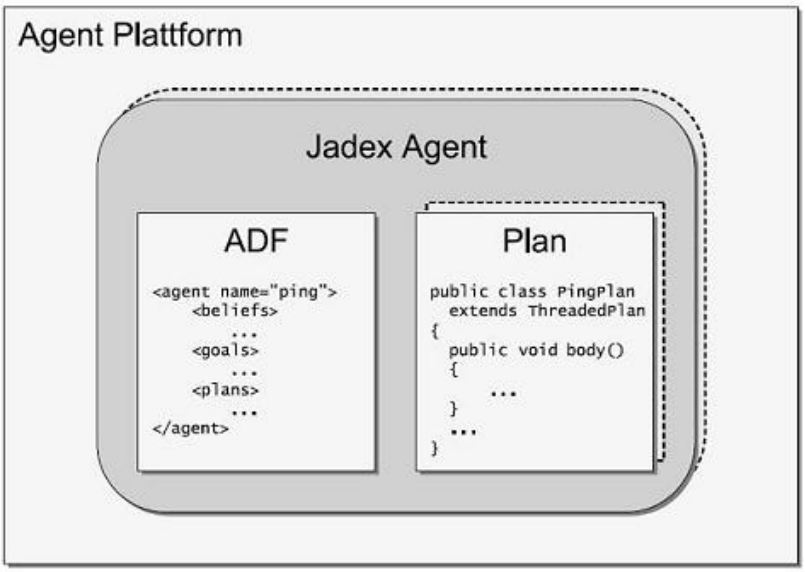

Figure 5.7: Components of a Jadex agent. [64]

schema[3]. The dynamic behavior (i.e. plan bodies) of an agent is specified in Java classes extending from a base class of the existing Jadex framework.

Jadex UML Profile

Following the Jadex meta-model (the XML schema), the author has created a Jadex UML profile in order to use UML constructs to model concepts defined in Jadex (Release 0.96), and thus to ease the transformation of the MAS PIM prepared using the AML profile into the Jadex PSM. The concepts defined in the Jadex meta-model are mapped one-to-one onto the model constructs defined in the Jadex UML profile. More specifically, each element defined in the Jadex XML schema is represented as a stereotype in the Jadex UML profile; each attribute of an element is represented as a tagged value. The Jadex conditions are specified as constraints in the Jadex UML profile. To illustrate the realization of the Jadex profile, several stereotypes and tagged values created to represent a part of the Jadex XML schema are described next.

The Jadex XML schema defines <belief> and <beliefset> elements that can be specified in the Agent Definition File to define a single valued belief and a multi-valued belief set respectively. A belief may have a default fact specified using the <fact> tag. Alternatively, a belief set may have a collection of initial facts specified using a set of <fact> tags or a <facts> tag when the number of

[3]http://vsis-www.informatik.uni-hamburg.de/projects/jadex/jadex-0.96x/schema/jadex-0.96.html#element_parameter_Link0452A910

Name	Type	Use	Annotation
name	xs:string	required	The name of the fact(s) contained in the belief
description	xs:string	optional	The elements optional description
exported	xs:string	optional	If an element is exported it can be referenced from an outer capability.
class	xs:string	required	The required (super) Java class of the fact objects that can be stored in the belief.
updaterate	xs:long	optional	For dynamic expressions the update rate defines in what intervals the fact will be re-evaluated.
transient	xs:boolean	optional	Transient beliefs are not retained when persisting or migrating an agent.

Table 5.2: The attributes of the belief element defined in the Jadex XML schema.

Figure 5.8: Use of Jadex profile to model a belief.

initial facts is not known in advance. In Table 5.2 is described the attributes of the belief element defined in the Jadex XML schema.

In Figure 5.8 is shown an example of applying the created Jadex profile to model a belief of an agent. The class *OperationCompleteBF* with stereotype $\ll belief \gg$ represents a Jadex belief. The attribute of the class *isCompleted* with stereotype $\ll fact \gg$ represents the fact of the belief that has a default value *new Boolean(false)*. The required attributes of the belief element are specified as tagged values: *{name=isCompleted, class=Boolean}*.

AML to Jadex Mapping

In order to successfully transform the MAS PIM into the Jadex PSM expressed in terms of the Jadex UML profile, the author has created the following mappings (names such as class names specified in the mapping tables are variables that are to be specified by developers):

Belief In Table 5.3 is presented the mappings between an AML belief and a Jadex belief. In Figure 5.9 is shown an example of using the

	AML Belief	Jadex Belief
(a)	Class named *BeliefName,* tagged with $\ll belief \gg$ has a single attribute whose type is not *Collection*	Class named *BeliefName,* tagged with $\ll belief \gg$, and the attribute is tagged with $\ll fact \gg$
(b)	Class named *BeliefName,* tagged with $\ll belief \gg$ has multiple attributes which have the same type other than *Collection*	Class named *BeliefName,* tagged with $\ll beliefset \gg$, and each of the attributes is tagged with $\ll fact \gg$
(c)	Class named *BeliefName,* tagged with $\ll belief \gg$ has a single attribute whose type is *Collection*	Class named *BeliefName,* tagged with $\ll beliefset \gg$, and the attribute is tagged with $\ll facts \gg$

Table 5.3: Mappings between an AML belief and a Jadex belief.

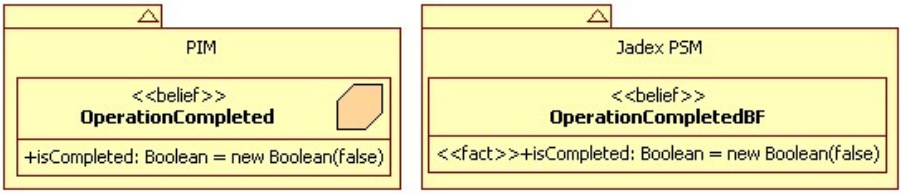

Figure 5.9: Transformation from an AML belief to a Jadex belief.

mapping (a) to transform an AML belief into a Jadex belief. The QiQu scripts used to implement this mapping are listed in Appendix C.1.

Goal In Table 5.4 is depicted the mappings between an AML goal and a Jadex goal. In Figure 5.10 is shown an example of using the mapping (a) and (b) to transform an AML goal into a Jadex achieve goal. The QiQu script used to implement this mapping are listed in Appendix C.2.

Plan In Table 5.5 is depicted the mappings between an AML plan and a Jadex plan. As the mappings used for plan transformations are very similar to the ones used for goal transformations, the QiQu script for the mapping implementation is not included.

Condition In Table 5.6 is presented the mappings between AML constraints and Jadex conditions.

An example is described here to illustrate how a combination of the above mappings are applied in this research to transform the MAS PIM created using

	AML Goal	**Jadex Goal**
(a)	Class named *GoalName,* tagged with ≪ *dgoal* ≫/≪ *ugoal* ≫ has a tagged value named *goalType e.g. {goalType=achievegoal}*	Class named *GoalName* and tagged with the value of the tagged value e.g. ≪ *achievegoal* ≫
(b)	Class named *GoalName,* tagged with ≪ *dgoal* ≫/≪ *ugoal* ≫ has an association with a class named *AssocEndClass* The association has a name *isParameterFor* and the association end at the side of *AssocEndClass* has a name *assocName*	Class *GoalParameterAssocName* is created in the Jadex PSM and tagged with ≪ *parameter* ≫, only if such a class does not already exist An association with the same properties of *isParameterFor* is created between the transformed goal class *GoalName* and *GoalParameterAssocName*

Table 5.4: Mappings between an AML goal and a Jadex goal.

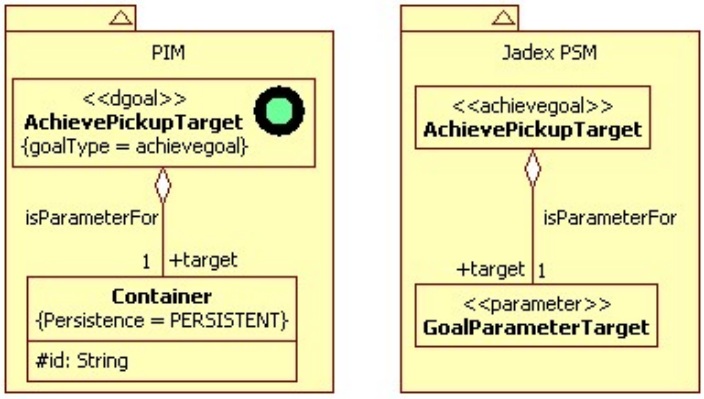

Figure 5.10: Transformation from an AML goal to a Jadex goal.

	AML Plan	**Jadex Plan**
(a)	Class named *PlanName,* tagged with ≪ *plan* ≫	Class named *PlanName,* tagged with ≪ *plan* ≫
(b)	Class named *PlanName,* tagged with ≪ *plan* ≫ has an association with a class named *AssocEndClass* The association has a name *isParameterFor* and the association end at the side of *AssocEndClass* has a name *assocName*	Class *PlanParameterAssocName* is created in the Jadex PSM and tagged with ≪ *parameter* ≫, only if such a class does not already exist An association with the same properties of *isParameterFor* is created between the transformed plan class *PlanName* and *PlanParameterAssocName*

Table 5.5: Mappings between an AML plan and a Jadex plan.

AML Constraints	Jadex Conditions
preCondition:≪ *pre* ≫	creationcondtion:≪ *creationcondition* ≫
commitCondition:≪ *commit* ≫	contextcondition:≪ *contextcondition* ≫
invariant:≪ *inv* ≫	maintaincondition:≪ *maintaincondition* ≫
cancelCondition:≪ *cancel* ≫	dropcondition:≪ *dropcondition* ≫
postCondition:≪ *post* ≫	targetcondition:≪ *targetcondition* ≫

Table 5.6: Mappings between AML constraints and Jadex conditions.

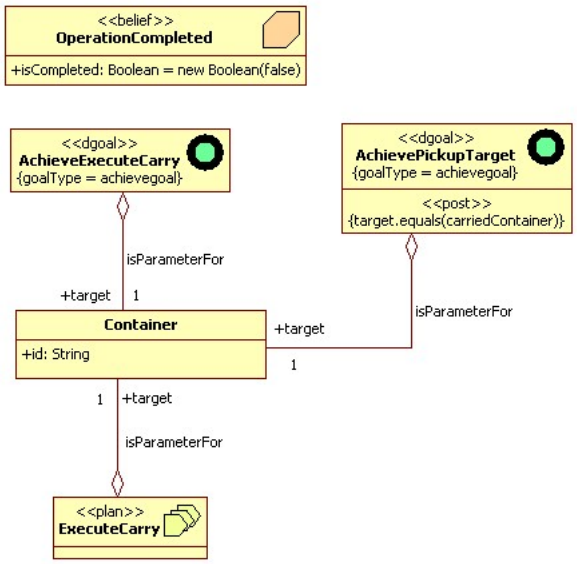

Figure 5.11: A part of the prototype MAS PIM.

the AML profile into the Jadex PSM. In Figure 5.11 is shown a small part of the prototype MAS PIM. By applying a set of mappings, the corresponding part of the Jadex PSM is generated and depicted in Figure 5.12. The stereotypes (e.g ≪ *unique* ≫, ≪ *trigger* ≫) and tagged values (e.g. *{class=Container}*) that are exclusively specific to the Jadex PSM were entered manually into the transformed PSM.

5.3.2 Platform Independent Model to JADE Platform Specific Model

In this section is described the process of transforming the MAS PIM into the JADE PSM. JADE is employed as an object persistence mechanism to store all information that is needed to help the MAS achieve its goal. As described in the previous section, Jadex uses Java for programming agents. Java and JADE are both object-oriented so that there is a natural fit between them. Furthermore,

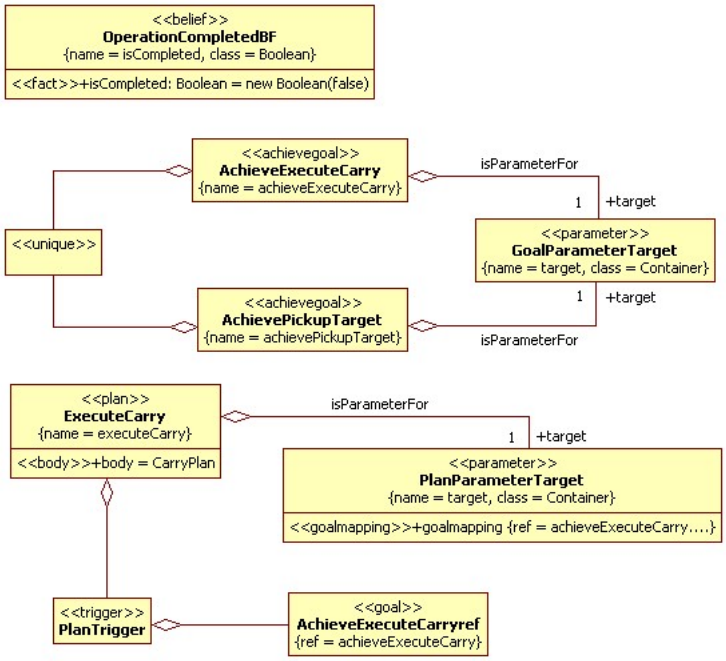

Figure 5.12: A part of the prototype MAS Jadex PSM.

Java objects can be persisted as JADE objects with none of the relational to object impedance mismatches found using a relational database. Thus, JADE is used in the prototype MAS to persist Java objects.

The Jade Software Corporation has developed a JADE-Java framework which provides all of the functionality required to add object persistence to a Java application. In order to persist a Java object in JADE, the persistence class of the object, must exist in both Java and JADE environments. In building the prototype MAS, the Java persistence classes are firstly developed and act as the proxies for the corresponding JADE persistence classes, which are generated automatically. More specifically, the approach consists of the following steps [79]:

1. Manually develop Java persistence classes and then apply the annotations (see Java annotations specification[4] for more information about annotations in Java 1.5) defined in the framework to mark the classes and their features.

2. Automatically generate application-related Java code required to use the annotated classes, such as Java code for connecting to the JADE system.

[4]http://www.jcp.org/en/jsr/detail?id=175

3. Use the Annotation Processor utility provided by the framework to automatically generate JADE schema files and forms definition files from the annotated Java classes so as to make the JADE persistence classes available.

4. Load the generated JADE schema files into JADE.

JADE Description

A number of JADE concepts applied in the prototype MAS to facilitate object persistence are described here (see [78] for a complete description of the JADE platform).

JADE is an object-oriented software development platform that provides object persistence as an integral part of the platform. A JADE object represents a real life entity that has a set of features and a unique identity. Features include properties (or data) whose values present the current state of an object and methods that describe the actions an object can perform. When an object is created, a unique object identifier (OID) is assigned by JADE to the object for distinguishing it from all others.

The set of objects that share the common characteristics (i.e. properties and methods) are described by a JADE class. JADE classes are arranged in a tree-like structure which represents inheritance ("is-a-kind-of") relationships among classes. A subclass inherits the features of its parent class (superclass) which is at the higher level of the inheritance hierarchy. The root of the class hierarchy is the *Object* class.

A JADE collection is an object that stores either values of the primitive types (e.g. a series of integers) or references to other objects (e.g. a series of object references). There are three main types of collections: (a) Set: an unordered collection of objects and can not store primitive type values; (b) Array: an ordered collection of objects or primitive values; and (c) Dictionary: an ordered collection of objects. A dictionary orders the objects based on user-defined key(s).

A schema, a collection of classes, is the highest-level organizational structure in JADE. Similar to the class hierarchy, schemas are arranged in the inheritance hierarchy as well. A subschema inherits all the classes (including associated properties and methods) that are defined in its superschema. The *RootSchema*, which is the root of the schema hierarchy, provides predefined essential system classes such as the *Object* class. JADE only permits single inheritance i.e. a subclass/ subschema only can have one direct superclass/superschema.

JADE UML Profile

Tag	Use	Documentation
name	optional	The name of the corresponding JADE class. The default name is the Java name.
defaultStyle	optional	Default persistence style for properties within the class. The value for this tag is to be one of: *DefaultStyle.NONE* *DefaultStyle.FIELD* *DefaultStyle.PROPERTY* *DefaultStyle.TRANSIENT.*
mapFile	optional	JADE schema map file. Defaults to the schema default map file.

Table 5.7: Tagged values for a Java persistence class.

Figure 5.13: An example of a persistence class.

The first step of persisting a Java object in JADE is to develop Java persistence classes and annotate them using Java annotations, as described earlier in this section. In the current annotation process, developers manually annotate Java persistence classes. In order to raise the design level from implementation to PSM so as to automate the annotation process, the author has created a JADE UML profile, which provides developers with UML constructs to model Java classes that are to be persisted in JADE. The JADE UML profile comprises a collection of stereotypes and tagged values representing the annotations and their required/optional attributes specified in the JADE-Java API Specification [80].

Class A Java class tagged with $\ll Entity \gg$ represents a persistence class, which is to be persisted in JADE. Three tagged values described in Table 5.7 may be specified to a persistence class. In Figure 5.13 is shown a persistence abstract class *CM_Object* with no tagged value specified.

Id Each persistence class must have a property of type *long* or *Long* tagged with $\ll Id \gg$. This field holds a unique value that links the Java objects of the persistence class with the corresponding persistent JADE objects. In Figure 5.13 is presented the *oid* field tagged with $\ll Id \gg$.

Tag	Use	Documentation
name	optional	Name of the corresponding JADE property. The default name is the Java name.
length	optional	Applies to properties which map to JADE Binary, Decimal, or String types.
scaled	optional	Applies only to Decimal.
type	optional	Allows an alternative JADE type to be declared.

Table 5.8: Tagged values for a Java persistence property.

Property There are two types properties:

- **Field**: A regular Java instance variable of primitive type, which has *get* (or *is* for a boolean value) and *set* methods, is tagged with $\ll DbField \gg$ for persistence.
- **Reference**: The stereotype $\ll DbProperty \gg$ is created for annotating a persistence reference property, where there are only get/is and *set* methods without there being a corresponding instance variable. The stereotype $\ll DbProperty \gg$ is applied to the *get/is* method.

Four tagged values described in Table 5.8 may be specified to a persistence property tagged with $\ll DbField \gg$ or $\ll DbProperty \gg$. In Figure 5.14 is shown a persistence class *CM_ Container* that has the following properties to be persisted: an instance field *id:String* and two reference properties whose *get/set* methods are *get/setFromLocation* and *get/setToLocation* respectively. The field *id* is tagged with $\ll DbField \gg$ and has a tagged value *{length=80}*. The *get* methods of the two reference properties are tagged with $\ll DbProperty \gg$.

Association In Table 5.9 is presented a set of stereotypes that are applied to reference properties in addition to $\ll DbProperty \gg$, for annotating different kinds of associations between two classes. A collection of tagged values, as described in Table 5.10, are associated with these stereotypes. As depicted in Figure 5.15, the method *getFromLocation* in the class *CM_ Container* and the method *getContainerAtFromLocation* in the class *CM_ Location* are tagged with $\ll DbProperty, OneToOne \gg$ to denote the two classes have a one-to-one association.

Collection The stereotype $\ll CollectionEntity \gg$ is created for annotating the specific collection classes which are required for collections of

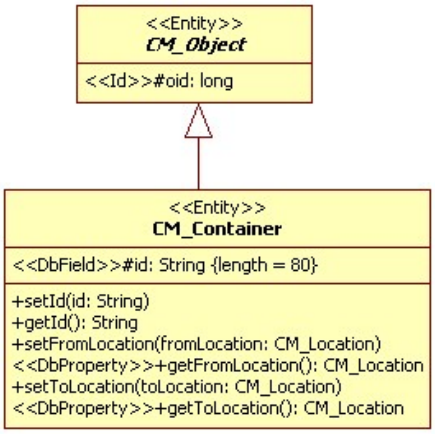

Figure 5.14: An example of marking persistence properties.

Stereotype	Documentation
$\ll OneToOne \gg$	To annotate the one-to-one relationship between two classes.
$\ll OneToMany \gg$	To annotate the one-to-many relationship between two classes.
$\ll ManyToOne \gg$	To annotate the many-to-one relationship between two classes.
$\ll ManyToMany \gg$	To annotate the many-to-many relationship between two classes.

Table 5.9: Stereotypes crated for annotating associations.

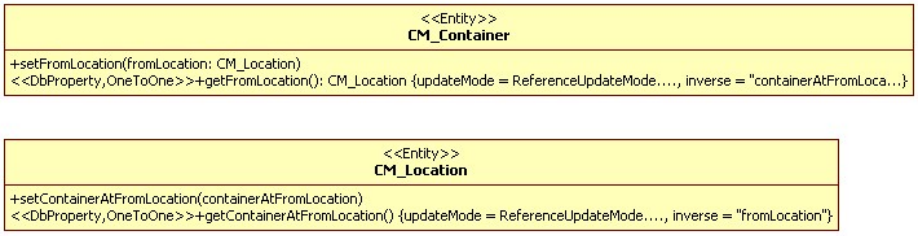

Figure 5.15: An example of annotated one-to-one relationship.

Tag	Use	Documentation
inverse	optional	The name of the inverse property in the referenced class.
relationshipType	optional	Whether there is a cascading deletion of referenced objects at one end of the relationship. The value for this tag is to be one of: *ReferenceRelationshipType.PARENT* *ReferenceRelationshipType.CHILD* *ReferenceRelationshipType.PEER*
updateMode	optional	Whether the property is updated manually in Java code or automatically by JADE. The value for this tag is to be one of: *ReferenceUpdateMode.AUTOMATIC* *ReferenceUpdateMode.MANUAL* *ReferenceUpdateMode.MAN_AUTO* *ReferenceUpdateMode.DEFAULT*
exclusive	optional	Applied to collection references only. Whether the collection is automatically created and deleted.
constraint	optional	Applied to collection references only. Name of a method that determines whether inverse maintenance is carried out.
transientToPersistentAllowed	optional	Whether the inverse maintenance from the persistent object to the transient is suppressed.
inverseNotRequired	optional	Whether an exception is raised when no inverse is set during automatic maintenance.

Table 5.10: Tagged values for persistence reference properties.

Tag	Use	Documentation
name	optional	The name of the corresponding JADE class. The default name is the Java name.
memberClass	optional	Java objects of this type populate the collection.
mapFile	optional	Defaults to the member class map file.

Table 5.11: Tagged values for a Java persistence collection.

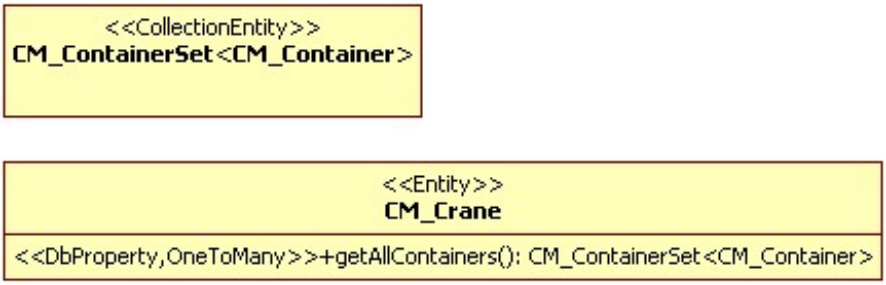

Figure 5.16: An example of annotated collection class.

Java persistence classes. Each annotated collection class must inherit from one of base collection classes provided by the JADE-Java framework (See [79] for details), to inherit the necessary behavior. As described in Table 5.11, three tagged values are associated with this stereotype. In Figure 5.16 is shown a collection class *CM_ ContainerSet* tagged with $\ll CollectionEntity \gg$.

Package A persistence package is a Java package containing one or more persistence classes. In each persistence package, a java class named *package-info* is included for annotation. The Java class *package-info* tagged with $\ll Schema \gg$ denotes that the package containing the annotated *package-info* class is a persistence package. As shown in Table 5.12, four tagged values are available for annotating the *package-info* class. An example of annotated *package-info* class in shown in Figure 5.17.

Figure 5.17: An example of annotated package-info class.

Tag	Use	Documentation
name	required	JADE schema name for the persistence classes.
applicationName	optional	Name of non-GUI JADE application to run when signing on to the schema. Defaults to the schema name.
superSchema	optional	Name of superschema of this JADE schema. Defaults to RootSchema (the top-most JADE schema).
defaultMapFile	optional	Default map file to use for persisted classes. Defaults to a map file with same name as the schema.

Table 5.12: Tagged values for a Java package that contains persistence classes.

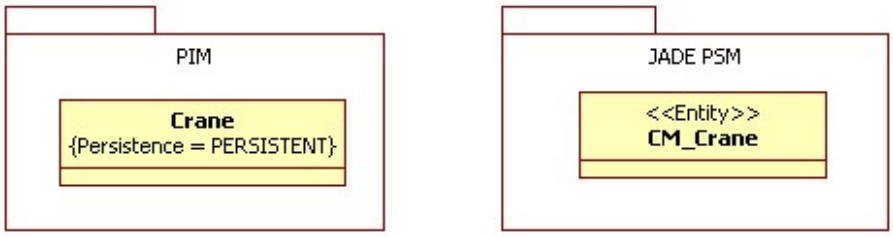

Figure 5.18: An example of transforming a PIM class into a JADE PSM class.

PIM to JADE Mapping

As described in Section 5.2.2, the tagged value *{Persistence=PERSISTENT}* is applied to annotate the PIM classes whose state are to be persisted. In order to transform those annotated PIM classes into JADE PSM classes, the author has defined a set of mappings.

Class For each class tagged with *{Persistence=PERSISTENT},* named *ClassName* in the PIM, a class named *CM_ ClassName* (the class name is changed only for name convention), tagged with $\ll Entity \gg$ is generated in the JADE PSM. In Figure 5.18 is shown an example of applying this mapping to transform an annotated PIM class into the corresponding annotated JADE PSM class.

Field For each public attribute (*public attributeName:Type*) of an annotated PIM class *ClassName*, a corresponding annotated attribute, its getter and setter methods are generated in JADE PSM class *CM_ ClassName* as:

- $\ll DbField \gg$ *protected attributeName:Type*

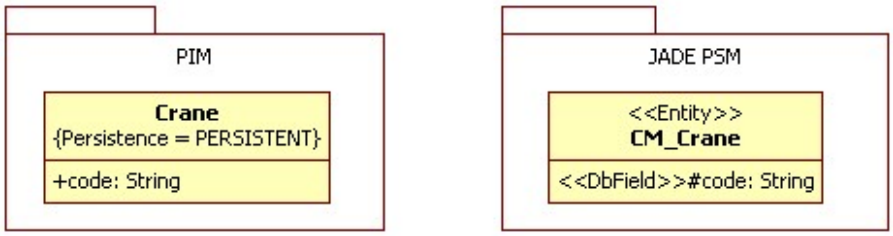

Figure 5.19: An example of class instance field transformation.

- A public getter *public Type getAttributeName()* (if *Type* is *Boolean* then *public Boolean isAttributeName()*)
- A pubic setter *public void setAttributeName(att : Type)*

In Figure 5.19 is presented an example of field transformation using the mapping.

Association For each association between *Class1* and *Class2* in the PIM:

- The association end at the side of *Class1* has a name *AssocEndName1* and a multiplicity *M1*
- The association end at the side of *Class2* has a name *AssocEndName2* and a multiplicity *M2*

transformation rules for the association end at the side of *Class1* with a multiplicity *M1* are:

- If $M1 = 1$ then in *CM_ Class2* create:
  - A public getter tagged with $\ll DbProperty, OneToOne \gg$ *public CM_ Class1 getAssocEndName1()*
  - A public setter *public void setAssocEndName1()*
- If $M1 > 1$ and the association end is declared as *UNORDERED* then:
  - In the package containing *CM_ Class1* create a class named *CM_ Class1Set* and tagged with $\ll CollectionEntity \gg$
  - In *CM_ Class2* create a public getter that is tagged with $\ll DbProperty, OneToMany \gg$ *public CM_ Class1Set getAssocEndName1()*
  - In *CM_ Class1* create:

        * A public getter tagged with $\ll DbProperty, ManyToOne \gg$ *public CM_ Class2 getAssocEndName2()*
        * A public setter *public void setAssocEndName2()*
    - If *M1* > 1 and the association end is declared as *ORDERED* then:
        - In the package containing *CM_ Class1* create a class named *CM_ Class1List* and tagged with $\ll CollectionEntity \gg$
        - In *CM_ Class2* create a public getter that is tagged with $\ll DbProperty, OneToMany \gg$ *public CM_ Class1List getAssocEndName1()*
        - In *CM_ Class1* create:
            * A public getter tagged with $\ll DbProperty, ManyToOne \gg$ *public CM_ Class2 getAssocEndName2()*
            * A public setter *public void setAssocEndName2()*

    By appropriately changing the variable names, a similar set of transformation rules are applied for the association end at the side of *Class2* with a multiplicity *M2*. In Figure 5.20 is presented an example of association transformation using the above rules.

An example is shown here to illustrate how a combination of the above mappings is applied to transform the PIM into the JADE PSM. In Figure 5.21 on the following page is shown a part of the prototype MAS PIM comprising three classes tagged with *{Persistence=PERSISTENT}*. By applying a combination of the above mappings, the JADE PSM is generated and depicted in Figure 5.22 on page 77. The abstract class *CM_ Object*, which defines the *id* field required for all the persistence classes, is created so that its subclasses do not need to separately define the field again. The stereotypes (e.g. $\ll Schema \gg$) and tagged values (e.g. *{defaultMapFile="ports"}*) that are exclusively specific to the JADE PSM were entered manually into the PSM.

5.4 Platform Specific Model to Implementation Transformation

The transformations from the Jadex PSM and the JADE PSM to their corresponding implementation models, as discussed in Sections 5.4.1 and 5.4.2 respectively, are performed in the third stage of the MDA process described in Section 5.1.3 on page 52. As depicted in Figure 5.23, a PSM in the format

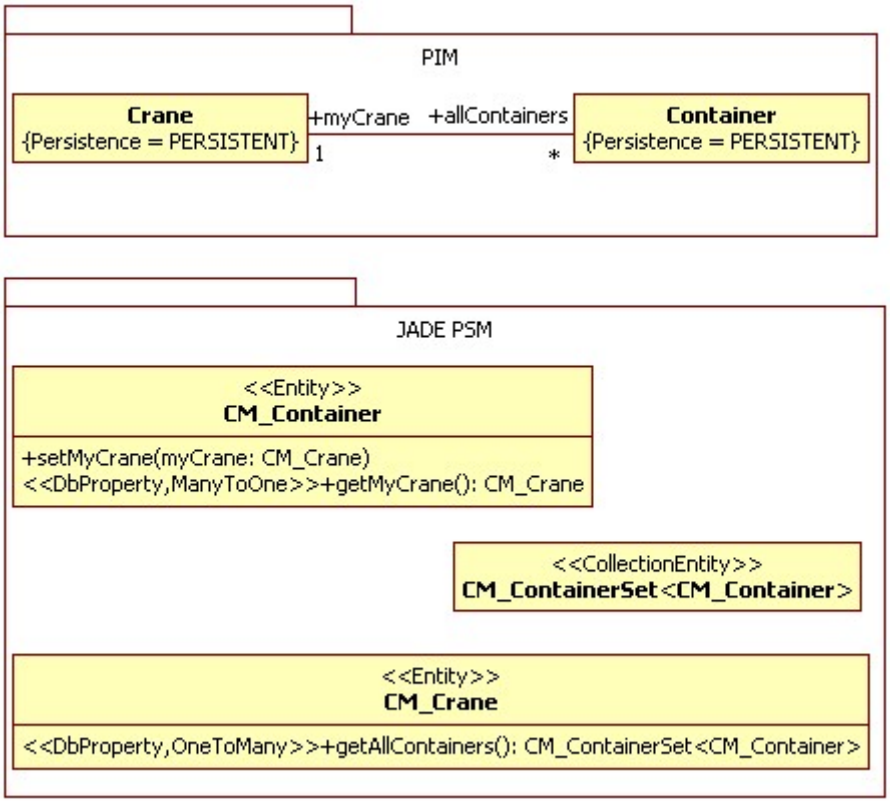

Figure 5.20: An example of one-to-many association transformation.

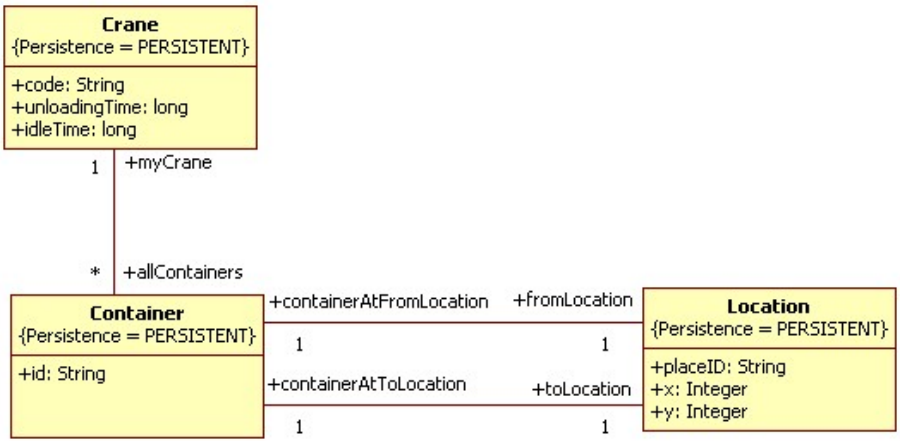

Figure 5.21: A part of the prototype MAS PIM.

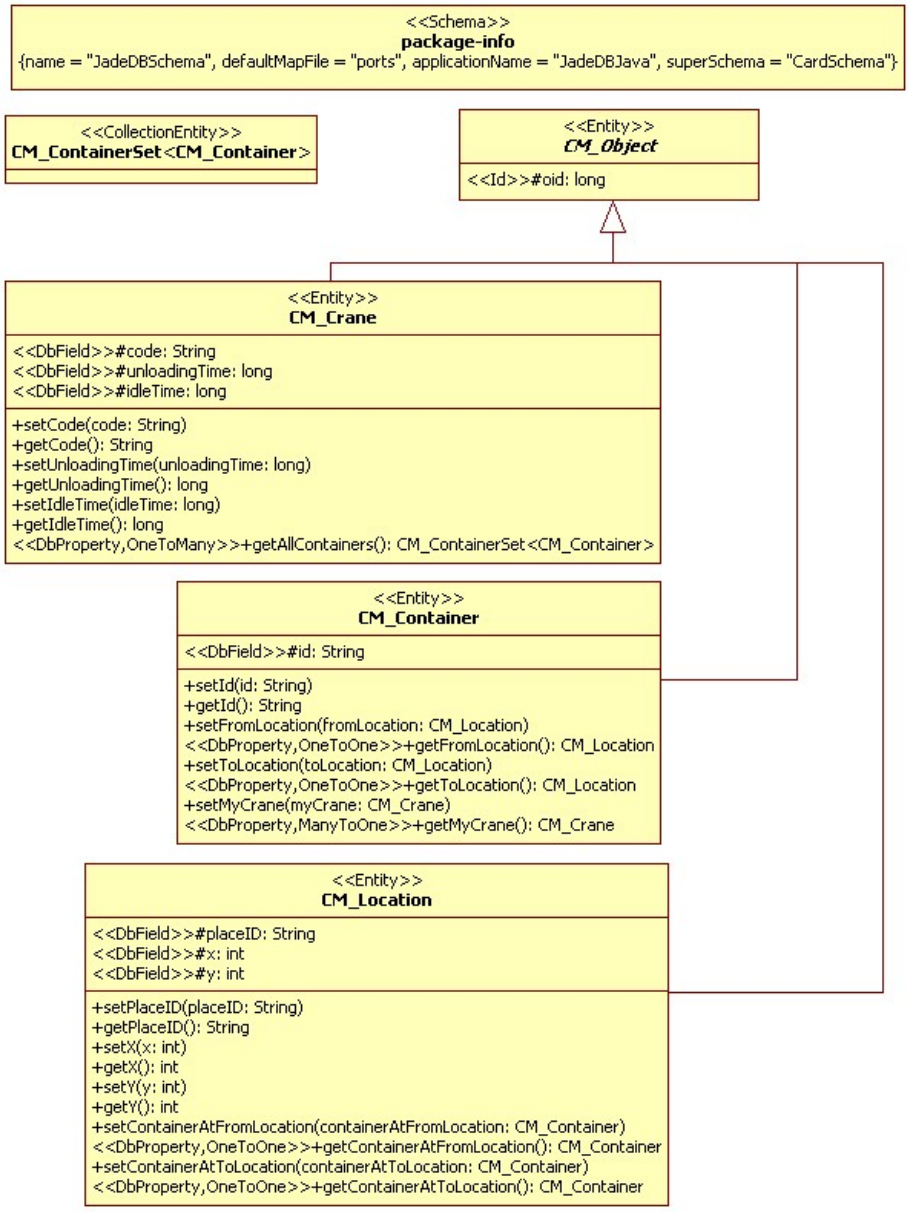

Figure 5.22: A part of the prototype MAS JADE PSM.

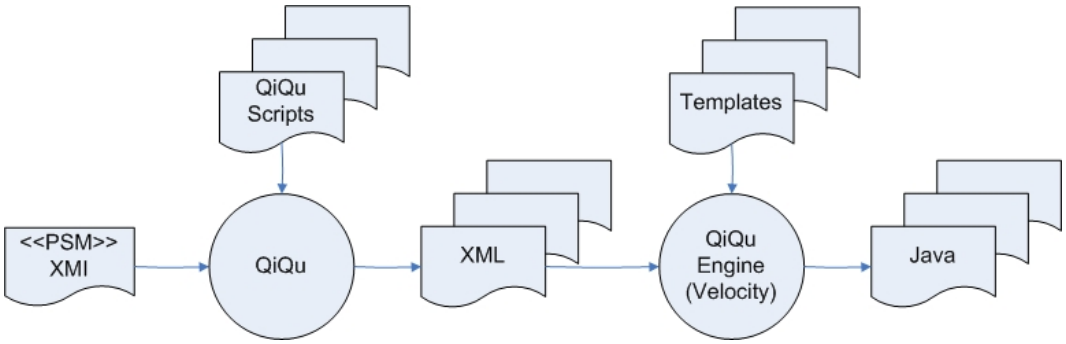

Figure 5.23: Transformation of a PSM into the corresponding implementation.

of XMI is transformed into the corresponding implementation according to the following steps:

1. The PSM in XMI is converted into an XML output-model which fulfills the requirements of QiQu-Velocity template engine. This step is implemented using QiQu scripting language. QiQu-Scripts allow developers to

   (a) Navigate over a UML model in XMI;

   (b) Select the elements of the source model; and

   (c) Transform the selected elements into the XML output-model, adding and/or removing information.

2. The transformation from the XML output-model to the desired implementation is done using the QiQu-Velocity template engine. The template engine merges the necessary templates defined in velocity template language and the XML output-model to produce the desired implementation (e.g. Java source code).

5.4.1 Jadex Platform Specific Model to Implementation

As presented in Section 5.3.1 on page 59, a Jadex agent's static structure is specified in an Agent Definition File, which is an XML file; the dynamic behavior of an agent is specified in Java classes extending from a base class of the existing Jadex framework.

The transformation from the Jadex PSM to Agent Definition Files is straightforward because the structures of the source model and the target model are already very similar. Several QiQu scripts for the implementation of the transformation rules are presented in Appendix D. Fragments of the generated Agent Definition File can be found in Appendix E.1.

The transformation from the Jadex PSM to Java implementation is used to realize plan bodies of a Jadex agent. As the elaborative MDA approach is adopted in this research (see Section 2.5 on page 22), the output implementation model of the transformation only consists of Java class skeletons. The method bodies of the transformed Java classes have to be specified manually in the source code.

5.4.2 JADE Platform Specific Model to Implementation

The approach of persisting Java objects in JADE, as presented in Section 5.3.2 on page 65, is to firstly develop annotated Java persistence classes and then automatically generate the corresponding JADE persistence classes using the Annotation Processor utility provided by the JADE-Java framework. Hence, the transformation from the JADE PSM to its corresponding implementation is actually the process of converting a UML model (in XMI) into a set of annotated Java classes. The author has developed a set of transformation rules for converting the JADE PSM into a simplified Java model, which holds the structure information of Java classes, namely the package reference, the attributes, the methods, and the annotations. As described in Section 5.3.2 on page 65, the stereotypes and tagged values provided by the JADE UML profile are representations of the annotations and their attributes specified in the JADE-Java framework. Thus, the realization of the transformations between them is very straightforward. The JADE PSM to implementation transformation rules are defined as follows:

- For each package named *packageName* in the JADE PSM, a Java package with the corresponding name, which contains the full path information delimited by dots ".", is created.
- For each JADE PSM class named *ClassName* in the package *packageName*, a Java class is created in the corresponding Java package with the following properties:
  - It has a package reference;
  - It has the same modifier as the class *ClassName*;
  - It has the same name as the class *ClassName*;
  - It extends the same super class as the class *ClassName*;
  - It implements the same set of interfaces as the class *ClassName*;
  - If the JADE PSM class *ClassName* is tagged with $\ll ClassStereotype \gg$ and one or more associated tagged values *{tag1=value1,tag2=value2}*,

then the corresponding Java class has the annotation *@ClassStereotype(tag1=value1,tag2=value2)*.

- For each attribute *attributeName* contained in the PSM class *ClassName*, a Java instance field is created in the corresponding Java class with the following properties:
  - It has the same modifier as the attribute *attributeName*;
  - It has the same type as the attribute *attributeName*;
  - It has the same name as the attribute *attributeName*;
  - If the attribute *attributeName* is tagged with $\ll FieldStereotype \gg$ and one or more associated tagged values *{tag1=value1,tag2=value2}*, then the corresponding Java instance field has the annotation: *@FieldStereotype(tag1=value1,tag2=value2)*.
- For each operation *operationName* contained in the PSM class *ClassName*, a Java method is created in the corresponding Java class with the following properties:
  - It has the same modifier as the operation *operationName* ;
  - It has the same return type as the operation *operationName* ;
  - It has the same name as the operation *operationName* ;
  - It has the same set of parameters as the operation *operationName* ;
  - If the operation *operationName* is tagged with one or more stereotypes $\ll PropertyStereotype, AssocStereotype \gg$ and a set of tagged values for each stereotype (*{tag1=value1}* for $\ll PropertyStereotype \gg$ and *{tag2=value2}* for $\ll AssocStereotype \gg$, then the corresponding Java method has the annotations: *@PropertyStereotype(tag1=value1)* and *@AssocStereotype(tag2=value2)*.

Fragments of the generated Java code can be found in Appendix E.2.

5.5 Interaction Bridge

In this section is described the generation of the interaction bridge between the Jadex PSM and the JADE PSM. In Figure 1.1 on page 3 is shown the MDA approach for developing the prototype MAS. The PIM comprises two components (database and business logic) each of which is under a separate control. As each component of the PIM may be transformed into a separate

PSM involving a different technology, an interaction bridge is needed to allow the distinct PSMs to interoperate.

In general, interoperability is defined as [36, p.42]: "The ability of two or more systems or components to exchange information and to use the information that has been exchanged." Specific to the research described here, the problem being addressed is the "exchange information" part of interoperability, the so called *syntactic interoperability* [37].

The MDA addresses this problem by generating from the PIM not only the two PSMs, but the necessary interaction bridge between them as well. The generated bridge is responsible for transforming and mapping the concepts from one implementation platform into the concepts used in another platform.

5.5.1 Interaction Bridge at the Platform Independent Model Level

The MDA improves model reusability by raising the software design level from implementation and PSMs to PIMs. When a change of implementation platform occurs, new PSMs together with bridges among them will be automatically generated without affecting the PIM. The automatic generation process is performed by executing a set of appropriate transformations, which have to be developed first if there are not existing yet. Abstracting the interoperability from the PSM level to the PIM level enables the same abstract design to be implemented multiple times using different technologies in an automatic fashion. In addition, the PIM is independent of any particular implementation technology and thus provides a clear view of the interoperability concern.

Within a service-based model paradigm such as that used in this research, an interaction occurs between a service requester and a service provider. A service is exposed via an interface. An interface, acting as a contract between a service provider and a service requester, defines the types of messages (operations) and data that are involved in the interaction between them. Thus, in order to successfully interact with a service provider, a service requester has to provide input data and receive output data in the specific types and formats (e.g. object models) defined by the service interface.

In this research, the PIM described in Section 5.2 on page 54 comprises two distinct components: database and agent business logic, which are to interact. The database component is the service provider delivering the persistence service for service requesters. The agent business logic component comprising a set of agents who act as the service requesters. The service-based approach allows service requesters to invoke the desired service delivered by the provider as an independent service in a standardized way such as web service [81] and CORBA

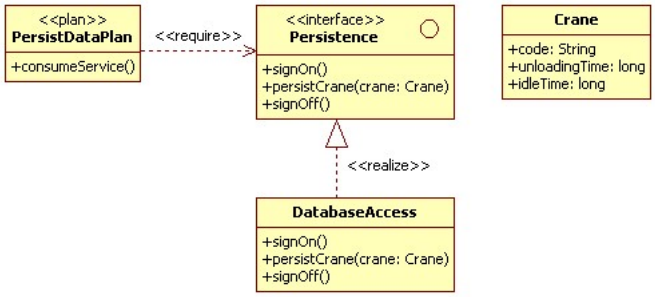

Figure 5.24: A service is exposed via an interface.

[82]. Hence, the service-based approach provides a standard and generic means of flexible interaction (i.e. exchange of information) among distinct components of the PIM that may be implemented on heterogeneous platforms.

In Figure 5.24 is shown a service requester *PersistDataPlan* and a service interface *Persistence*, which is realized by *DatabaseAccess*. Furthermore, in the figure is also shown that he dependency between the service requester and the service interface is tagged with $\ll require \gg$.

5.5.2 Interaction Bridge at the Platform Specific Model Level

In this research, the Jadex is selected for the development of the business logic component of the PIM. JADE is employed as an object persistence mechanism. Thus, JADE is the service provider that delivers the persistence service to Jadex agents.

Use of the JADE-Java framework, as described in Section 5.3.2 on page 65, enables Jadex agents to persist Java objects in JADE. In order to use this framework, the classes of the objects to be persisted need to exist in both Java and JADE environments, as shown in Figure 5.25. The transformation from the MAS PIM to the JADE PSM described in Section 5.3.2 results in a set of annotated Java classes acting as the proxies for the corresponding JADE persistence classes.

In Figure 5.26 is described the interaction bridge at the PSM level. The service interface *Persistence* is realized by the class *DatabaseAccess*, which defines a simple entry point to access the objects defined by the JADE PSM. Moreover, a *DatabaseAccess* object acts as the intermediary between service requesters and the service provider. Service requesters communicate with the provider by sending requests to the *DatabaseAccess* object, which then forwards them to the appropriate object(s) defined by the JADE PSM. In addition, *DatabaseAc-*

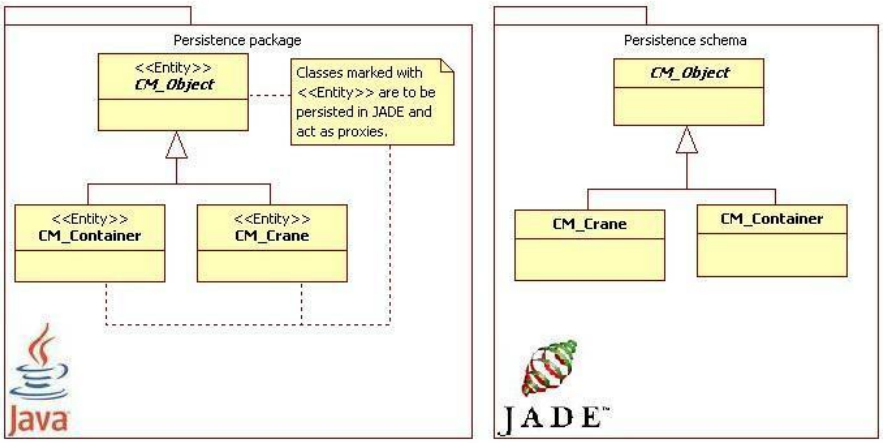

Figure 5.25: The persistence classes exist in both Java and JADE environments.

cess may handle mapping/transformation services to allow communications with the provider. Use of a service interface decouples the service provider from requesters, so that varying the implementation of the provider will not affect its requesters.

5.5.3 Interaction Bridge at the Implementation Level

In this section is described the generation of the interaction bridge implementation from the UML model described in the previous section. The classes modeled in the class diagram are mapped one-to-one onto a set of annotated Java classes in the implementation. The generation of the implementation of the Java proxies is already described in Section 5.4.2 on page 79. Once the annotated Java proxy classes are generated, the corresponding JADE persistence classes can then be generated automatically from those Java proxy classes using the Annotation Processor utility provided by the JADE-Java framework (see Section 5.3.2 on page 65). In Appendix E.3 can be found some fragments of the generated code.

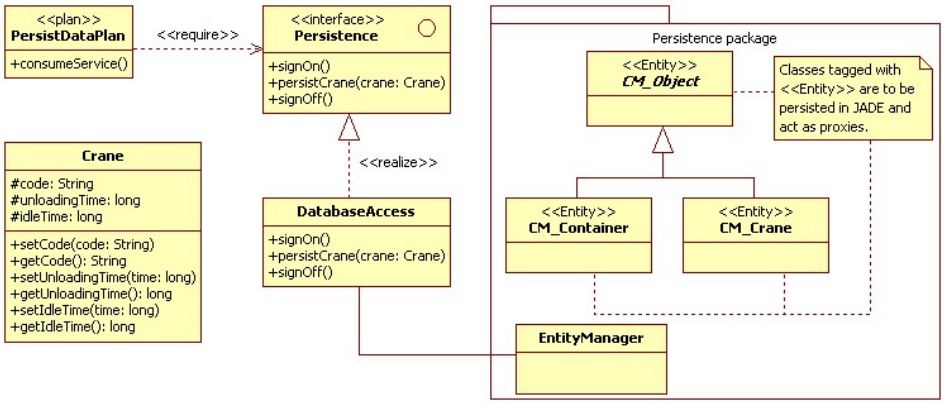

Figure 5.26: Interaction bridge at the PSM level

Chapter 6

Conclusions

In the last chapter of the thesis, the author concludes the performed research by presenting a summary, the answers to the research questions listed in Section 1.2 on page 4, and possible future research.

6.1 Summary

In this thesis is presented the use of MDA to develop a prototype MAS to support the management of a real container terminal. As an outcome of the research, a prototype MAS has been developed for evaluating and comparing the performance of the given set of vehicle dispatching strategies in the container discharging process. As described in Section 2.4.1.2 on page 14, container terminal is a highly dynamic environment within which there are a large number of diverse and independent industry entities whose individual decisions may directly affect the performance of the others. It is very difficult or impossible to build a single centralized system to fulfill the requirements of container terminals due to their distributed nature. Moreover, the survey of Paul et al. [5] described in Section 2.4.3 on page 20 suggests that agent technology is a promising approach to solve a wide range of distributed and complex problems within transport logistics, in particular container terminals. A MAS offers the following three advantages compared with a single system that has a centralized control: (a) ability to distribute control; (b) ability to cope with incomplete and uncertain data; and (c) ability to model a complex environment.

Experiments were carried out in order to test the applicability of the MAS in the application domain of logistics. More specifically, the prototype MAS operates off-line to compare the performance of the proposed SC dispatching strategies in a variety of real-world scenarios that vary in terms of ship sizes and numbers of QCs. The experiment results have revealed that MASs are

applicable to assist container terminal decision makers in evaluating operating strategies.

In this research, the MDA has been applied to semi-automatically derive the MAS implementation from the platform independent agent-oriented design. The author's main contribution is the implementation of the MDA process, which is composed of three stages: (1) Manually creating the MAS PIM; (2) Generating a set of target PSMs and the necessary interaction bridges among them from the MAS PIM using a set of mappings; and (3) The PSMs and the interaction bridges among them are automatically transformed into the implementation.

The author's experience of using the MDA for the development of the prototype MAS has revealed that the MDA approach results in the following advantages compared to the traditional approach. (a) Automated transformations between models increase software productivity; (b) Separating the high-level specification of the system from the underlying implementation technology improves the portability of the system's high-level abstraction model. (c) Strong separation of concerns, guaranteed consistency between models, and automatic generation of source code minimize future software maintenance effort.

6.2 Answers to Research Questions

In this section is presented the answers to the research questions listed in Section 1.2 on page 4.

The first research question is:

> *What motivates the use of MDA in the development of software systems?*

Today, a majority of complex software systems are developed using the traditional code-centric approach. Nevertheless, developers face a number of problems such as development productivity, component portability/reusability, system interoperability and maintenance. The MDA is a new software development paradigm that is able to address these problems. In contrast to the traditional code-centric software development, the MDA based software development uses models as the primary engineering artifacts. The MDA facilitates software development by raising the software design level from source code to the platform independent model and automating the transformations from the high-level abstraction model into the corresponding implementation. More details regarding the answer to this question are presented in Section 2.5.1 on page 23.

The second research question, which is the main research question of the thesis, is:

How to apply the MDA to agent technologies, providing a partially automated support for the derivation of MAS implementation from the agent-oriented design, independently from the target implementation platforms?

The above research question is decomposed into the following subquestions:

How to define the high-level abstraction model of the MAS?

The high-level abstraction model (PIM) of the MAS was manually defined by applying the AML profile to describe the system in an agent-oriented level. AML was chosen as the modeling language for developing the MAS PIM because it is an implementation platform independent language for specifying and modeling MASs. The MAS PIM created using AML is a high-level abstraction model that is portable to different agent implementing platforms. Moreover, AML is MOF compliant and supported by UML CASE tools so that XMI could be used during the MDA process to facilitate the MAS development. The answer to this research question is described in more detail in Section 5.2 on page 54.

How to transform the high-level abstraction model of the MAS into a set of implementation technology specific models?

The MAS PIM comprises two components, business logic and database, each of which was transformed into a separate PSM expressed using a different platform specific UML profile. Jadex has been selected for the development of goal-oriented agents. JADE has been employed as an object persistence mechanism. Model transformations from the MAS PIM into the PSMs were realized by:

1. Executing a collection of mappings defined using QiQu scripting language. Firstly, the author has created two platform specific UML profiles i.e. the Jadex and JADE UML profiles. Then, the author has created two sets of mappings for transforming the MAS PIM prepared using the AML profile into the Jadex and JADE PSMs expressed in terms of their corresponding UML profiles respectively.

2. Elaborating the PSMs, automatically generated from the above step, by manually inserting the model constructs such as stereotypes and tagged values that are exclusively specific to each of the PSMs.

The formal definition of the PIM-to-PSM transformations for the prototype MAS are described in Section 5.3 on page 58. The formal definition is quite complex and is a time-consuming task to develop it. However, without applying MDA, these kinds of transformations have always been executed by hand and

not automated nor well defined. After the formalization is completed, it can be reused many times in many projects and the payback can be huge.

> *How to transform the set of implementation technology specific models into the corresponding implementation?*

The author has developed a collection of transformation rules using a combination of QiQu scripting language and Velocity templates for transforming each of the two refined PSMs into a separate implementation. The implementation was generated in an automatic fashion because of the fact that the PSMs and the corresponding implementation are relatively close to each other, and have almost the same level of abstraction. The PSM-to-Implementation transformations are presented in Section 5.4.

> *How to enable the distinct implementation technology specific models to interoperate despite differences in programming languages and execution platforms?*

As the two PSMs generated from the PIM target different implementation platforms, they cannot directly interact with each other. This has created a need for interoperability. In Section 5.5 is described how the MDA addresses this problem by generating from the PIM not only the two PSMs, but the necessary interaction bridge between them as well. The generated bridge is responsible for transforming and mapping the concepts from one implementation platform into the concepts used in another platform.

6.3 Future Research

From the user point of view, one disadvantage of the prototype MAS is that it does not consider the dynamic, real-time nature of the domain, e.g. breakdown of equipment, weather conditions, or the congestion on the road. One option for the future research to deal with the stochastic nature of the container terminal is to use a real-time MAS to manage the distributed and independent processes. A real-time MAS is a multi-agent system that operates in a time-critical environment [1]. This paradigm may be an appropriate solution to container terminal management, which requires distributed decision making as well as rapid responses [2, 3, 4]. The JMT stores all the real-time information on container flows and various available resources. Therefore, it can provide sufficient real-time input data for a real-time MAS.

From the developer point of view, the adopted MDA approach has its limits. Through the development of the prototype MAS presented in this thesis, it can

be noted that the MDA process does not provide 100% of the final implementation. This is because the system's dynamic behavior needs to be specified manually in the source code. Thus, with the aim of increasing the percentage of automation in the MDA process for future research, the focus may be shifted to capture the system's dynamic behavior in the PIM and/or PSMs.

Another possible future research is to enhance the transformation tool used in this research by adding additional features such as tunability, model validation and bidirectionality [22]. Tunability refers to the ability to tune/change the transformation definitions according to the users specification. For example, when transforming a public attribute of a PIM class to a protected attribute of the target PSM class with getter and setter methods, the methods' prefix strings (usually get and set) can be specified by the user. Model validation refers to the model checking ability such as syntax/semantics checking, consistency checking. Bidirectionality refers to the ability to perform transformations from both directions, i.e. not only from source to target, but also back from target to source.

Appendix A

Interview Summary

According to the interview with the operation manager and the ship controller at Port Otago, a number of issues have been identified as having an impact on the working of the ship, as well as a number of functionalities that are desirable to meet future challenges are summarized as follows:

- Hatch Lid Placement
  - Hatch lids can be placed on the ground or on adjacent hatch on the ship. If placed on an adjacent hatch then this may interfere with a subsequent job.
  - It would be useful if the system could record where a hatch lid has been placed, and warn if this is likely to cause subsequent problems such as:
    * The hatch lid blocking access to the hold it has been placed on.
    * Containers having to be loaded on top of the area where the hatch lid has been placed.
- Reefer Disconnections
  - Reefer containers have to be disconnected prior to being loaded on the ship.
  - It would be useful if the system could alert the controller, based on the current work rate, ahead of time so they can ensure that the reefers are about to be worked. This is particularly critical for chilled cargo as it can only be off power for 30 minutes.

- Overstowed Containers
  - Overstowed containers in the yard create additional work and delays when loading to the ship. Efforts are made when planning and sequencing containers to the ship to reduce the number of overstow moves required. If cranes work at different rates to the predicted, or the order of jobs is swapped then the system needs to be able to quickly calculate the impact this will have on the number of overstow moves.
- Rehandles and Transships
  - Rehandles are containers that are being discharged and then loaded elsewhere onto the same ship. Transships are containers that are being discharged off one ship and loaded onto another ship.
  - The system needs to warn if a rehandle or transship container is not going to be discharged before it is to be loaded.
- Crane Clashing
  - When multiple cranes are working a ship they have to keep a minimum distance apart. This is because physically the cranes cannot be that close and as well as to allow for access of cargo handling equipment such as straddle carriers.
  - Care is taken when planning the ship to avoid crane clashes but as work progresses the system needs to monitor for any potential crane clashes and warn the operator as soon as possible so remedial action can be taken. The remedial action may involve re-ordering jobs or reallocating jobs to another crane. In either case this has implications for the sequencing of the cargo to the ship.
- Over-Dimensional Containers
  - Over-dimensional containers are those that are a non-standard size and/or have cargo that extends beyond the normal dimensions of a standard container. They require special equipment to lift on or off the ship, which requires time to prepare and setup. It can take over 1 hour to load or discharge an over-dimensional container.
  - The system needs to be able to give a warning to the operator a configurable amount of time before an over-dimensional container is due to be loaded or discharged so that adequate preparation can be done.

- Resource Allocation
  - The decision of how many machines should be assigned to a QC can greatly affect the crane productivity. Good crane intensity (i.e. all working cranes have the same operating time) is desirable. Inefficient number of machines assigned to serve a crane may delay the ship completion time which results in the next ship scheduled for berthing has to wait. In such a case, the terminal company normally needs to pay a fine (thousands dollars).
- It is desirable that if the system can calculate and track the number of lifts for each crane working on a ship.

Appendix B

Experiment Input Example

The following shows an example of experiment input data.

```
CM_Crane,C2,
CM_Container,TRLU6063045,101,90,C2TA 3,B 3913 1,
CM_Container,PONU2867085,166,3,C2TA 4,C 2611 1,
CM_Container,MWCU5608806,71,7,C2TA 3,C 1911 2,
CM_Container,MAEU5694473,137,53,C2TA 4,C 2610 1,
CM_Container,MAEU5694241,49,100,C2TA 3,C 1305 1,
CM_Container,APMU2827486,230,121,C2TA 3,F 1504 3,
CM_Container,SAMU2208642,18,30,C2TA 3,F 1501 1,
CM_Container,MSKU3603169,87,120,C2TA 4,B 3613 2,
CM_Container,PONU0324094,1255,4,C2TA 4,F 1505 2,
CM_Container,PONU4981374,71,30,C2TA 4,C 2315 1,
CM_Container,PONU4818710,79,4,C2TA 3,A 2307 2,

CM_Crane,C3,
CM_Container,TTNU1515544,167,312,C3TA 1,F 1811 1,
CM_Container,MSKU3693265,304,163,C3TA 1,K 3401 1,
CM_Container,PONU0983532,73,168,C3TA 1,K 1701 1,
CM_Container,MSKU2776318,295,143,C3TA 1,K 1604 1,
CM_Container,CRXU1682916,310,193,C3TA 1,F 1714 1,
CM_Container,POCU0278966,38,124,C3TA 1,F 1418 1,
CM_Container,MSKU2758802,197,147,C3TA 1,F 1614 1,
CM_Container,TRLU3511594,67,215,C3TA 1,F 1712 1,
CM_Container,PONU0766434,283,246,C3TA 1,F 1810 1,
CM_Container,MSKU2147193,79,95,C3TA 1,F 1713 1,
CM_Container,FBXU8889770,100,166,C3TA 1,F 1711 1,
CM_Container,MSKU2321577,28,206,C3TA 1,F 1412 2,

CM_StraddleCarrier,9,23,17,12,20,11,22,25,24,21,
```

Appendix C

Platform-Independent Model to Platform-Specific Model Transformations

C.1 Transformations from an AML belief to a Jadex belief

The following QiQu scripts implement the mapping (a), as described in Table 5.3 on page 63, for transforming an AML belief to a Jadex belief.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<QiQuScript>
  <SelectFirst NodeRef="$agent"
      XPath=".//UML:Stereotype[@name='belief']"
      SelectedEleRef="$beliefStereotype"/>
    <If Condition="exists($beliefStereotype)">
      <ForString Input="$beliefStereotype.extendedElement"
              Delimiter="' '" IteratorRef="beliefClassID">
        <SelectFirst NodeRef="$agent"
            XPath=".//UML:Class[@xmi.id='#beliefClassID#']"
            SelectedEleRef="$beliefClass"/>
        <Set Ref="counter" Value="'0'"/>
        <For NodeRef="$beliefClass"
            XPath=".//UML:Attribute" IteratorEleRef="$attribute">
            <Set Ref="counter" Value="addition(counter , '1')"/>
        </For>
        <If Condition="equals(counter,'1')">
          <SelectFirst NodeRef="$beliefClass"
            XPath=".//UML:Attribute" SelectedEleRef="$fact"/>
          <SelectFirst NodeRef="$outModel"
            XPath=".//UML:DataType[@xmi.id='#$fact.type#']"
            SelectedEleRef="$factType"/>
          <If Condition="not(equals($factType.name , 'Collection'))">
            <LoadDoc FileName="jadex + 'Fact.qiq'"
                    NewDocRef="$script"/>
            <RunQiQuScript NodeRef="$script"/>
          </If>
        </If>
      </ForString>
    </If>
</QiQuScript>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<QiQuScript>
  <SelectFirst NodeRef="$agent"
    XPath=".//UML:Stereotype[@name='fact']"
    SelectedEleRef="$factType"/>
  <If Condition="not(exists($factType))">
    <CreateEle NodeRef="$namespace"
        EleName="'UML:Stereotype'"
        NewEleRef="$factType"/>
    <Set Ref="curID" Value="addition(curID , '1')"/>
    <Set Ref="$factType.xmi.id" Value="'X.'+ curID"/>
    <Set Ref="$factType.name" Value="'fact'"/>
  </If>
  <If Condition="not(exists($factType.extendedElement))">
    <Set Ref="$factType.extendedElement" Value="''"/>
  </If>
  <For NodeRef="$beliefClass" XPath=".//UML:Attribute"
                              IteratorEleRef="$fact">
    <Set Ref="$factType.extendedElement"
        Value="trim($factType.extendedElement + ' ' + $fact.xmi.id)"/>
  </For>
</QiQuScript>
```

C.2 Transformations from an AML goal to a Jadex goal

The following QiQu script provides a partial implementation of the mapping (a) and (b), as described in Table 5.4 on page 64, for transforming an AML goal to a Jadex achieve goal.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<QiQuScript>
  <SelectFirst NodeRef="$agent"
    XPath=".//UML:Namespace.ownedElement"
    SelectedEleRef="$namespace"/>
  <SelectFirst NodeRef="$agent"
    XPath=".//UML:Stereotype[@name='dgoal']"
    SelectedEleRef="$goalStereotype"/>
  <If Condition="exists($goalStereotype)">
    <ForString Input="$goalStereotype.extendedElement"
        Delimiter="' '" IteratorRef="ClassID">
      <LoadDoc FileName="jadex + 'GoalType.qiq'" NewDocRef="$script"/>
      <RunQiQuScript NodeRef="$script"/>
      <SelectFirst NodeRef="$agent"
        XPath=".//UML:Class[@xmi.id='#ClassID#']"
        SelectedEleRef="$goalClass"/>
      <If Condition="exists($goalClass)">
        <If Condition="exists($goalClass.participant)">
          <ForString Input="$goalClass.participant"
            Delimiter="' '" IteratorRef="associationID">
            <SelectFirst NodeRef="$agent"
                XPath=".//UML:AssociationEnd[@xmi.id='#associationID#']"
                SelectedEleRef="$associationEnd"/>
            <SelectFirst NodeRef="$associationEnd" XPath=".."
                SelectedEleRef="$association"/>
            <SelectFirst NodeRef="$association"
                XPath=".//UML:AssociationEnd[1]"
                SelectedEleRef="$associationEnd1"/>
            <LoadDoc FileName="jadex + 'AssociationParameter.qiq'"
                NewDocRef="$script"/>
            <RunQiQuScript NodeRef="$script"/>
          </ForString>
        </If>
      </If>
    </ForString>
  </If>
</QiQuScript>
```

Appendix D

Platform-Specific Model to Code Transformations

The following QiQu scripts provide a partial implementation for transforming a agent belief defined in the Jadex PSM to implementation.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<QiQuScript>
  <!-- create belief -->
  <SelectFirst NodeRef="$agent" XPath=".//UML:Stereotype[@name='belief']"
      SelectedEleRef="$beliefStereotype"/>
  <If Condition="exists($beliefStereotype)">
    <CreateEle NodeRef="$agentEle" EleName="'beliefs'" NewEleRef="$beliefsEle"/>
    <ForString Input="$beliefStereotype.extendedElement" Delimiter="' '"
        IteratorRef="beliefClassID">
      <SelectFirst NodeRef="$agent" XPath=".//UML:Class[@xmi.id='#beliefClassID#']"
          SelectedEleRef="$beliefClass"/>
      <If Condition="exists($beliefClass)">
        <CreateEle NodeRef="$beliefsEle" EleName="'belief'" NewEleRef="$beliefEle"/>
        <LoadDoc FileName="elePath + 'Belief.qiq'" NewDocRef="$script"/>
        <RunQiQuScript NodeRef="$script"/>
      </If>
    </ForString>
  </If>
  <!-- cretae beliefset -->
  <SelectFirst NodeRef="$agent" XPath=".//UML:Stereotype[@name='beliefset']"
    SelectedEleRef="$beliefsetStereotype"/>
  <If Condition="exists($beliefsetStereotype)">
    <If Condition="not(exists($beliefsEle))">
      <CreateEle NodeRef="$agentEle" EleName="'beliefs'" NewEleRef="$beliefsEle"/>
    </If>
    <ForString Input="$beliefsetStereotype.extendedElement" Delimiter="' '"
        IteratorRef="beliefsetClassID">
      <SelectFirst NodeRef="$agent" XPath=".//UML:Class[@xmi.id='#beliefsetClassID#']"
        SelectedEleRef="$beliefsetClass"/>
      <If Condition="exists($beliefsetClass)">
        <CreateEle NodeRef="$beliefsEle" EleName="'beliefset'" NewEleRef="$beliefsetEle"/>
        <LoadDoc FileName="elePath + 'Beliefset.qiq'" NewDocRef="$script"/>
        <RunQiQuScript NodeRef="$script"/>
      </If>
    </ForString>
  </If>
</QiQuScript>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<QiQuScript>
  <!-- set belief attributes -->
  <LoadDoc FileName="attPath + 'BeliefAttrFrag.qiq'" NewDocRef="$script"/>
  <RunQiQuScript NodeRef="$script"/>
  <!-- create fact -->
  <SelectFirst NodeRef="$agent" XPath=".//UML:Stereotype[@name='fact']"
        SelectedEleRef="$factStereotype"/>
  <If Condition="exists($factStereotype)">
    <ForString Input="$factStereotype.extendedElement" Delimiter="' '"
        IteratorRef="factAttrID">
      <SelectFirst NodeRef="$beliefClass"
        XPath=".//UML:Attribute[@xmi.id='#factAttrID#']"
        SelectedEleRef="$fact"/>
      <If Condition="exists($fact)">
        <SelectFirst NodeRef="$fact" XPath=".//UML:Expression"
            SelectedEleRef="$factBody"/>
        <If Condition="exists($factBody)">
          <CreateEle NodeRef="$beliefEle" EleName="'fact'"
            NewEleRef="$factEle"/>
          <LoadDoc FileName="elePath + 'Fact.qiq'" NewDocRef="$script"/>
          <RunQiQuScript NodeRef="$script"/>
        </If>
      </If>
    </ForString>
  </If>
</QiQuScript>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<QiQuScript>
<If Condition="exists($requirement)">
  <ForString Input="$requirement.value" Delimiter="' '" IteratorRef="attrit">
    <ForString Input="attrit" Delimiter="'='" IteratorRef="attr"/>
    <If Condition="not(contains(attr ,'[]' ))">
        <Set Ref="attName" Value="replace(attrit, '='+attr, '')"/>
    </If>
    <If Condition="contains(attr ,'[]' )">
        <Set Ref="attr" Value="replace(attr, '\\['+'\\]', '')"/>
        <Set Ref="attName" Value="replace(attrit, '='+attr+'\\['+'\\]', '')"/>
        <Set Ref="attr" Value="attr + '[]'"/>
    </If>
    <If Condition="equals(attName,'name')">
        <Set Ref="$beliefEle.name" Value="attr"/>
    </If>
    <If Condition="equals(attName,'description')">
        <Set Ref="$beliefEle.description" Value="attr"/>
    </If>
    <If Condition="equals(attName,'exported')">
        <Set Ref="$beliefEle.exported" Value="attr"/>
    </If>
    <If Condition="equals(attName,'class')">
        <Set Ref="$beliefEle.class" Value="attr"/>
    </If>
    <If Condition="equals(attName,'updaterate')">
        <Set Ref="$beliefEle.updaterate" Value="attr"/>
    </If>
    <If Condition="equals(attName,'transient')">
        <Set Ref="$beliefEle.transient" Value="attr"/>
    </If>
  </ForString>
</If>
</QiQuScript>
```

Appendix E

Transformed Source Code

E.1 Transformed Jadex Code Fragments

The following code snippet describes the Agent Definition File fragment transformed from the part of the Jadex PSM depicted in Figure 5.12.

```
<belief name="isCompleted" class="Boolean">
  <fact>new Boolean(false)</fact>
</belief>

<achievegoal name="achieveExecuteCarry">
  <parameter name="target" class="Container"/>
  <unique/>
</achievegoal>

<achievegoal name="achievePickupTarget">
  <parameter name="target" class="Container"/>
  <unique/>
  <targetcondition>
     $goal.target.equals($beliefbase.carriedContainer)
  </targetcondition>
  <failurecondition>
     $goal.target == null
  </failurecondition>
</achievegoal>

<plan name="executeCarry">
  <parameter name="target" class="Container">
     <goalmapping ref="achieveExecuteCarry.target"/>
  </parameter>
  <unique/>
</plan>
```

E.2 Transformed Annotated Java Code Fragments

The following code snippet describes the *package-info.java*.

```
@Schema(name="JadeDBSchema",
        superSchema="CardSchema",
        applicationName="JadeDBJava",
        defaultMapFile="ports")
package JadeDBSchema;
import com.jadeworld.jade.persistence.*;
```

The following code snippet describes the *CM_ Object.java*.

```
package JadeDBSchema;

import com.jadeworld.jade.entitymanager.EntityAccess;

@Entity()
public abstract class CM_Object {
    @Id
    protected long oid;

}
```

The following code snippet describes the *CM_ ContainerSet.java*.

```
package JadeDBSchema;

import com.jadeworld.jade.entitymanager.EntityAccess;

@CollectionEntity()
public class CM_ContainerSet<CM_Container> extends
 ObjectSet<CM_Container>{
}
```

The following code snippet describes the *CM_ Container.java*.

```
package JadeDBSchema;

import com.jadeworld.jade.entitymanager.EntityAccess;

@Entity()
public class CM_Container extends CM_Object{
    @DbField()
    protected String id;
    public void setId(String id){
        this.id = id;
    }
    public String getId(){
        return this.id;
    }

    public void setFromLocation(CM_Location fromLocation){
        EntityAccess.setReferenceProperty(this, "fromLocation",
 fromLocation);
    }
    @DbProperty()
    @OneToOne(relationshipType=ReferenceRelationshipType.PEER,
            updateMode=ReferenceUpdateMode.MANUAL,
            inverse="containerAtFromLocation")
    public CM_Location getFromLocation(){
        return(CM_Location)EntityAccess.getReferenceProperty(this,
 "fromLocation");
    }

    public void setToLocation(CM_Location toLocation){
        EntityAccess.setReferenceProperty(this, "toLocation",
 toLocation);
    }
    @DbProperty()
    @OneToOne(relationshipType=ReferenceRelationshipType.PEER,
            updateMode=ReferenceUpdateMode.MANUAL,
            inverse="containerAtToLocation")
    public CM_Location getToLocation(){
        return(CM_Location)EntityAccess.getReferenceProperty(this,
 "toLocation");
    }

    public void setMyCrane(CM_Crane myCrane){
        EntityAccess.setReferenceProperty(this, "myCrane", myCrane);
    }
    @DbProperty()
    @ManyToOne(relationshipType=ReferenceRelationshipType.CHILD,
            updateMode=ReferenceUpdateMode.MANUAL,
            inverses={"allContainers"})
    public CM_Crane getMyCrane(){
        return(CM_Crane)EntityAccess.getReferenceProperty(this,
 "myCrane");
    }
}
```

E.3 Interaction Bridge Code Fragments

The following code snippet describes the facade factory *JADEFacadeFactory.java*.

```
package bridge;

public class JADEFacadeFactory implements PersistenceFacadeFactory{
    public Persistence getDatabaseAccess(){
        return DatabaseAccess.getInstance();
    }
}
```

The following code snippet describes the service facade *DatabaseAccess.java*.

```
package bridge;

import port.Crane;

public class DatabaseAccess implements bridge.Persistence {
    private EntityManagerFactory factory;
    private EntityManager manager;
    private EntityTransaction transaction;
    private static String persistenceUnitName = "JadeDBSchemaPU";
    private static DatabaseAccess instance;

    public void signOn() {
        factory = Persistence
                .createEntityManagerFactory(persistenceUnitName);
        manager = factory.createEntityManager();
        transaction = manager.getTransaction();
    }

    public void persistCrane(Crane crane) {
        transaction.begin();
        CM_Crane cm_crane = new CM_Crane();
        manager.persist(cm_crane);
        cm_crane.setCode(crane.getCode());
        cm_crane.setIdleTime(crane.getIdleTime());
        cm_crane.setUnloadingTime(crane.getUnloadingTime());
        transaction.commit();
    }

    public void signOff() {
        factory.close();
    }

    public static bridge.Persistence getInstance() {
        if (instance == null) {
            synchronized (DatabaseAccess.class) {
                instance = new DatabaseAccess();
            }
        }
        return instance;
    }

    protected DatabaseAccess() {
        signOn();
    }
}
```

Bibliography

[1] J. Stankovic. Distributed real-time computing: The next generation. Journal of the Society of Instrument and Control Engineers of Japan, 1992.

[2] J. Soler, V. Julian, M. Rebollo, C. Carrascosa, and V. Botti. Towards a real-time MAS architecture. In Proceedings of Challenges in Open Agent Systems. AAMAS'02, Bolonia, Italia, 2002.

[3] V. Julian, C. Carrascosa, M. Rebollo, J. Soler, and V. Botti. Simba: an Approach for Real-Time Multi-Agent Systems. In Proceedings of V Conferncia Catalana d'Intel.ligncia Articial, Castell. Springer-Verlag, 2002.

[4] P. Stone and M. Veloso. Multiagent Systems: A Survey from a Machine Learing Perspective. Under review for journal publication February, 1997.

[5] Paul Davidsson, Lawrence Henesey, Linda Ramstedt, Johanna Törnquist and Fredrik Wernstedt. An analysis of agent-based approaches to transport logistics. Transportation Research Part C: Emerging Technologies, Volume 13, Issue 4, August 2005, Pages 255-271.

[6] Dariusz Barbucha and Piotr Je¸drzejowicz. An Agent-Based Approach to Vehicle Routing Problem. International Journal of Applied Mathematics and Computer Science, vol. 4 nr 2, pp. 538 – 543, 2007.

[7] Miguel Rebollo, Vicente Julian, Carlos Carrascosa and Vicente Botti. A Multi-Agent System for the Automation of a Port Container Terminal. Autonomous Agents 2000 workshop on Agents in Industry.

[8] Miguel Rebollo, Vicente Julian, Carlos Carrascosa and Vicente Botti. A MAS Approach for Port Container Terminal Management. 3rd Iberoamerican workshop on DAI-MAS, pp. 83-94, 2001

[9] OMG Unified Modelling Language Specification. OMG group, (http://www.omg.org/uml).

[10] http://www.jadeworld.com/jade/index.htm

[11] http://www.omg.org/technology/documents/profile_catalog.htm

[12] R. Cervenka and I. Trencansky. Agent Modeling Language: Language Specification. Version 0.9. Technical report, Whitestein Technologies, 2004.

[13] R. Cervenka, I. Trencansky, M. Calisti, and D. Greenwood. AML: Agent Modeling Language. Toward Industry-Grade Agent-Based Modeling. In J. Odell, P. Giorgini, and J.P. Muller, editors, Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004, pages 31-46. Springer-Verlag, Berlin, 2005.

[14] Ivan Trencansky and Radovan Cervenka. Agent modeling language (AML): A comprehensive approach to modeling MAS. Informatica, 29:391400, 2005.

[15] OMG. UML 2.0 OCL Specification. ptc/03-10-14, October 2003.

[16] J. Odell, H. Parunak, and B. Bauer. Extending UML for agents. G. Wagner, Y. Lesperance, and E. Yu, (eds.), Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, TX, 2000, pp. 3-17.

[17] B. Bauer, J. P. Muller, J. Odell. Agent UML: A Formalism for Specifying Multiagent Software Systems. International Journal of Software Engineering and Knowledge Engineering, vol.11, n.3, 2001. pp 207-230.

[18] Jishnu Mukerji, Joaquin Miller. Technical Guide to Model Driven Architecture: The MDA Guide v1.0.1. omg/2003-06-01. (http://www.omg.org/docs/omg/03-06-01.pdf)

[19] http://staruml.sourceforge.net/en/

[20] http://galaxy.andromda.org/index.php?option=com_content&task=blogcategory&id=0&Itemid=42

[21] http://www.aloba.ch/qiqu/home_en.html

[22] Anneke Kleppe, Jos Warmer, and Wim Bast. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 032119442X.

[23] Alexander Pokahr, Lars Braubach, Winfried Lamersdorf. Jadex: A BDI Reasoning Engine, Chapter of Multi-Agent Programming, Kluwer Book, Editors: R. Bordini, M. Dastani, J. Dix and A. Seghrouchni. Available at http://vsis-www.informatik.uni-hamburg.de/projects/jadex/papers/promasbook_jadex_revised.pdf

[24] M. Wooldridge. Agent-based software engineering. IEE Proc Software Engineering, vol. 144, pp. 26-37, 1997.

[25] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96), volume 1193, Berlin, Germany, 1996. Springer-Verlag.

[26] Michael Woolbridge and Nicholas R. Jennings. Agent Theories, Architectures,and Languages: a Survey. Intelligent Agents. Springer-Verlag, 1-22, 1995.

[27] http://www.jadeworld.com/logistics/index.htm

[28] Anne V. Goodchild, Carlos F. Daganzo, Crane Double Cycling in Container Ports: Effect on Ship Dwell Time, Institute of Transportation Studies University of California at Berkeley, July 2005.

[29] K. H. Kim, Park, Y.-M. A crane scheduling method for port container terminals. European Journal of Operational Research, vol. 156, pp. 752-768, 2004.

[30] J. Böse, Reiners, T., Steenken, D., and Voss, S. Vehicle Dispatching at Seaport Container Terminals Using Evolutionary Algorithms. presented at 33rd (HICSS -2000), Hawaii International Conference on System Sciences-Volume 2, Maui, Hawaii, US, 2000.

[31] Pietro Canonaco, Pasquale Legato, Rina M. Mazza and Roberto Musmanno, A queuing network model for the management of berth crane operations, Computers & Operations Research, In Press, Corrected Proof, Available online 5 February 2007,

[32] Parunak, H.V.D., Industrial and Practical Applications of DAI, In Multiagent Systems: A Modern Approach to Distributed Artifical Intelligence, edited by G. Weiss, pp. 377-421, MIT Press, Cambridge, MA, 1999

[33] Lawrence Henesey, Dissertation: Multi-Agent Container Terminal Management, Karlshamn. Blekinge Institute of Technology, 2006

[34] Larry Henesey, Fredrik Wernstedt, Paul Davidsson, Market-Driven Control in Container Terminal Management, Blekinge Institute of Technology Ronneby/ Sweden

[35] J.L. Poza, R. Simarro, M.A. de la Fuente, J. Simo. A. Crespo, REAL-TIME CONTROL AND MONITORING IN A CONTAINER TERMINADL, 15th Triennial World Congress, Barcelona, Spain 2002.

[36] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

[37] Grace A. Lewis Lutz Wrage. Approaches to Constructive Interoperability. TECHNICAL REPORT CMU/SEI-2004-TR-020 ESC-TR-2004-020, December 2004.

[38] Mellor, Stephen J., and Marc J. Balcer. Executable UML: A Foundation for Model-Driven Architecture. Boston: Addison Wesley, 2002.

[39] Kap Hwan Kim and Ki Young Kim. Routing straddle carriers for the loading operation of containers using a beam search algorithm, Computers and Industrial Engineering, v.36 n.1, p.109-136, Jan. 1999.

[40] Egbelu, P.J., Tanchoco, J.M.A., 1984. Characterization of automatic guided vehicle dispatching rules. International Journal of Production Research 22 (3), 359–374.

[41] René (M.)B. M. de Koster, Tuan Le-Anh and J. Robert van der Meer. Testing and classifying vehicle dispatching rules in three real-world settings. Journal of Operations Management, Volume 22, Issue 4, August 2004, Pages 369-386.

[42] Srinivasan, M.M., Bozer, Y.A., Cho, M. Trip-based material handling systems: Throughput capacity analysis. IIE Transactions 26 (1), 70–89, 1994.

[43] Le-Anh, T. and Koster, M.B.M. de René, Online Dispatching Rules for Vehicle-Based Internal Transport Systems (21 2004, 06). ERIM Report Series Reference No. ERS-2004-031-LIS. Available at SSRN: http://ssrn.com/abstract=594970

[44] Klein, C.M. and Kim, J. AGV dispatching. International Journal of Production Research, 34(1), 95-110, 1996.

[45] Jeong, B.H. and Randhawa, S.U. A multi-attribute dispatching rule for automated guided vehicle systems. International Journal of Production Research, 39(13), 2817-2832, 2001.

[46] Bozer, Y.A. and Yen, C. Intelligent dispatching rules for trip-based material handling systems. Journal of Manufacturing Systems, 15(4), 226-239, 1996.

[47] V. Graudina and J. Grundspenkis. Technologies and multi-agent system architectures for transportation and logistics support: An overview. International Conference on Computer Systems and Technologies - CompSysTech, 2005.

[48] Akio Imai, Etsuko Nishimura and Stratos Papadimitriou. The dynamic berth allocation problem for a container port. Transportation Research Part B: Methodological, Volume 35, Issue 4, May 2001, Pages 401-417.

[49] Pengfei Zhou, Haigui Kang, and Li Lin. Intelligent Control and Automation. A Dynamic Berth Allocation Model Based on Stochastic Consideration. The Sixth World Congress on Volume 2, 21-23 June 2006 Page(s):7297 - 7301.

[50] L. Moccia, J.-F. Cordeau, M. Gaudioso, and G. Laporte. A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal. Naval Research Logistics 53 (1), pp. 45-59, 2006.

[51] Grunow M., Günther H.O., and Lehmann, M.. Strategies for dispatching AGVs at automated seaport container terminals. OR Spectrum, vol. 28, pp. 587-610, 2006.

[52] L. Henesey. A Multi Agent Based Simulator for Managing a Container Terminal. 2nd European Workshop on Multi-Agent Systems (EUMAS 2004), December 16 - 17, 2004, Barcelona, Spain.

[53] C. Degano and A. Pellegrino. Multi-Agent Coordination and Collaboration for Control and Optimization Strategies in an Intermodal Container Terminal. Engineering Management Conference (IEMC-2002) IEEE International , Vol. 2, 18-20 Aug. 2002 Page(s):590 - 595.

[54] Rao, AS. and Georgeff, M. BDI agents: from theory to practice. Proceedings of the first international conference on Multi-agent systems, (ICMAS-95), San Francisco, CA, 1995, pp. 31 2-319.

[55] M. Bratman. Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, Massachusetts, 1987.

[56] Genesereth, M.R. and Ketchpel, S.P. Software Agents. Communications of the ACM, 37(7): pp. 48–53, 1994.

[57] Russell, S.J. and Norvig, P. Artificial Intelligence: a Modern Approach, 2nd edn. Prentice Hall, 2003.

[58] Rao A. S. and Georgeff M. P. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning. Morgan Kaufmann Publishers, San Mateo, CA, 1991, pp. 473-484.

[59] d'Inverno, M., Kinny, D., Luck, M. and Wooldridge, M. A Formal Specification of dMARS. In Singh, M.P., Rao, A.S. and Wooldridge, M. (eds), Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages, pp. 155–176, Springer, 1998.

[60] Huber, M. JAM: A BDI-Theoretic Mobile Agent Architecture. In Proceedings of the 3rd International Conference on Autonomous Agents, pp. 236–243, New York, NY, 1999.

[61] Howden, N., Ronnquist, R., Hodgson, A. and Lucas, A. JACK Intelligent Agents – Summary of an Agent Infrastructure. In Proceedings of the 5th International Conference on Autonomous Agents, 2001.

[62] M. Wooldridge. Intelligent Agents In G. Weiss, editor: Multiagent Systems, The MIT Press, April 1999. Available at http://www.csc.liv.ac.uk/~mjw/pubs/mas99.pdf

[63] Wooldridge, M. Reasoning about Rational Agents. The MIT Press, 2000.

[64] Alexander Pokahr and Lars Braubach. Jadex User Guide. Release 0.96, 1. June 2007. Distributed Systems Group University of Hamburg, Germany. Available at http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

[65] Fabio Luigi Bellifemine, Giovanni Caire, Dominic Greenwood. Developing Multi-Agent Systems with JADE. WILEY series i agent technology, ISBN: 978-0-470-05747-6, April 2007.

[66] Shoham Y. Agent-oriented programming. D. G. Bobrow. Artificial Intelligence Volume 60. Elsevier. Amsterdam. 1993. pp.51-92.

[67] Hindriks K., de Boer F., van der Hoek W., and Meyer J.-J. Agent Programming in 3APL. N. Jennings, K. Sycara, and M. Georgeff. Autonomous Agents and Multi-Agent Systems. Kluwer Academic publishers. 1999. pp. 357-401.

[68] Lehman J. F., Laird J. E., and Rosenbloom P. S. A gentle introduction to Soar, an architecture for human cognition. Invitation to Cognitive Science Vol. 4. MIT press. 1996.

[69] Lars Braubach, Alexander Pokahr, Daniel Moldt, Winfried Lamersdorf. Goal Representation for BDI Agent Systems. The Second International Workshop on Programming Multiagent Systems (PROMAS-2004).

[70] Ilaria Vacca, Michel Bierlaire, and Matteo Salani. Optimization at Container Terminals: Status, Trends and Perspectives. 7^{th}Swiss Transport Research Conference Monte Verita/Ascona, September 12.-14. 2007.

[71] http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

[72] FIPA Specification Repository web page. http://www.fipa.org/repository/

[73] Ivan Trencanský, Radovan Cervenka, Dominic A. P. Greenwood. Applying a UML-based agent modeling language to the autonomic computing domain. OOPSLA Companion 2006. pp.521-529.

[74] Vis, Iris F.A. Bakker, Maurice. Dispatching and layout rules at an automated container terminal. Vrije Universiteit Amsterdam 2005.

[75] Chin-I Liu and Petros Ioannou. A Comparison of Different AGV Dispatching Rules in an Automated Container Terminal. The IEEE 5th International Conference on Intelligent Transportation Systems, 2002. Page(s): 880 - 885.

[76] Meersmans, Patrick J. M. and Wagelmans, Albert P. M. Dynamic Scheduling of Handling Equipment at Automated Container Terminals. (13 2001, 12). ERIM Report Series Reference No. ERS-2001-69-LIS. Available at SSRN: http://ssrn.com/abstract=370931

[77] Morreale, V. Bonura, S. Francaviglia, G. Centineo, F. Cossentino, and M. Gaglio, S. Goal-Oriented Development of BDI Agents: The PRACTION-IST Approach. Intelligent Agent Technology, 2006. IAT '06. Page(s): 66-72.

[78] JADE Development Environment User's Guide. http://www.jadeworld.com/downloads/jade/manuals/UserGuide.pdf

[79] JADE Java Developer's Reference. http://www.jadeworld.com/downloads/jade/manuals/JavaDev.pdf

[80] JADE Java API Specification. http://www.jadeworld.com/downloads/jade/manuals/javajom-doc/index.html

[81] Web Services Architecture. W3C Working Group Note, W3C Web Services Architecture Working Group http://www.w3.org/TR/2004/ NOTE-ws-arch-20040211/wsa.pdf, 11 February 2004.

[82] Common Object Request Broker Architecture: Core Specification. http://www.omg.org/docs/ formal/04-03-01.pdf, March 2004.

[83] OMG UML Specification Version 1.3. ftp://ftp.omg.org/pub/docs/ad/99-06-08.pdf

[84] OMG XML Metadata Interchange (XMI) Specification Version 1.1. http://www.omg.org/docs/ad/99-10-02.pdf

[85] Java Emitter Templates Version 2. http://www-128.ibm.com/developerworks/opensource/library/os-ecl-jet/

[86] XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/1999/REC-xpath-19991116

[87] Dragan Gaˇsevi´ and Dragan Djuri´. Model Driven Architecture and Ontology Development. Springer Berlin Heidelberg. 2006. ISBN 978-3-540-32180-4 (Print) 978-3-540-32182-8 (Online).

[88] OMG, Meta Object Facility (MOF) 2.0 Core Specification. Version 20, 2003. Available at: http://www.omg.org/docs/ptc/03-10-04.pdf. Accessed in: 11/2004.

[89] FIPA: Foundation for Intelligent Physical Agents Geneva, Switzerland 1997

[90] Cervenka, Trencansky. Agent Modeling Language Language Specification Version 0.9. Whitestein 2004-12-20.